

Reference Guide

VERSION

2.0

Borland[®]
ObjectWindows[®]
for C++

Reference Guide

Borland
ObjectWindows[®]
for C++
Version 2.0

Borland may have patents and/or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

COPYRIGHT © 1991, 1993 by Borland International. All rights reserved. All Borland products are trademarks or registered trademarks of Borland International, Inc. ObjectWindows is a registered trademark of Borland International, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

Borland International, Inc.

100 Borland Way, Scotts Valley, CA 95067-3249

PRINTED IN THE UNITED STATES OF AMERICA

1E0R993
9394959697-9876543

W1

Contents

Introduction	1	IDW_MDIFIRSTCHILD constant	26
Contents of this manual	1	ImParent constant	26
Icons and typefaces used in this manual	2	LongMulDiv function	26
Conventions used in this manual	3	MAX_RSRC_ERROR_STRING constant	26
ObjectWindows hierarchy diagram	3	NBits function	27
Chapter 1 Library reference	7	NColors function	27
TBird class [sample]	8	ofxxxx document open enum	27
Public data members	8	pfxxxx property attribute constants	28
Public constructors and destructor	8	_BUILDOWDLL macro	28
Public member functions	8	_OWLCCLASS macro	29
Protected data members	9	_OWLDATA macro	29
Protected constructors	9	_OWLDLL macro	29
Protected member functions	9	_OWLFAR macro	29
Response table entries	10	_OWLFUNC macro	29
ObjectWindows libraries	10	OWLGetVersion function	30
The ObjectWindows header files	11	shxxxx document sharing enum	30
The ObjectWindows resource files	15	TActionFunc typedef	30
The ObjectWindows library reference	16	TActionMemFunc typedef	30
BF_xxxx constants	16	TAnyPMF typedef	31
CM_xxxx edit constants	16	TAnyDispatcher typedef	31
CM_xxxx edit file constants	17	TApplication class	31
CM_xxxx edit file exit constant	17	Public data members	31
CM_xxxx edit replace constants	17	Public constructors and destructor	32
CM_xxxx MDI constants	18	Public member functions	33
DECLARE_RESPONSE_TABLE macro	18	Protected data members	37
DEFINE_DOC_TEMPLATE_CLASS macro	18	Protected member functions	38
DEFINE_RESPONSE_TABLE macros	19	TApplication::TXInvalidMainWindow class	39
dmxxxx document manager mode constants	19	Public constructors	39
dnxxxx document message enum	20	Public member functions	39
dt document view constants	20	TBandInfo class	40
END_RESPONSE_TABLE macro	21	TBitmap class	40
EV_xxxx macros	21	Public constructors	40
ID_xxxx file constants	22	Public member functions	42
ID_xxxx printer constants	23	Protected member functions	43
IDA_xxxx accelerator ID constants	23	TBitmapGadget class	43
IDM_xxxx menu ID constant	23	Public constructors and destructor	44
IDS_xxxx document string ID constants	23	Public member functions	44
IDS_xxxx edit file ID constants	24	Protected member functions	44
IDS_xxxx exception messages	24	TBitSet class	45
IDS_xxxx listview ID constants	25	Constructors	45
IDS_xxxx printer string ID constants	25	Public member functions	45
IDS_xxxx validator ID constants	25	TBrush class	46
IDW_MDICLIENT constant	26	Public data members	46
		Protected data members	47

Public constructors	47	TClipboardViewer class	71
Public member functions	48	Public constructors	71
TButton class	48	Public member functions	71
Public data members	49	Protected data members	72
Public constructors	49	Response table entries	72
Protected data members	49	TColor class	72
Protected member functions	49	Public data members	72
Response table entries	50	Public constructors	73
TButtonGadget class	50	Public member functions	74
Public data members	51	Protected data members	75
Public constructors and destructor	51	TComboBox class	76
Public member functions	51	Public data members	76
Protected data members	52	Public constructors	76
Protected member functions	53	Public member functions	77
TCelArray class	55	Protected member functions	80
Public constructors and destructor	55	TComboBoxData class	81
Public member functions	56	Public data members	81
Protected data members	57	Public constructors and destructor	81
TCharSet class	57	Public member functions	81
Public constructors	57	TCommonDialog class	82
Public member functions	58	Public constructors	82
TCheckBox class	58	Public member functions	82
Public data members	58	Protected data members	82
Public constructors	58	Protected member functions	83
Public member functions	59	Response table entries	83
Protected member functions	60	TCondFunc type	83
Response table entries	61	TCondMemFunc typedef	84
TChooseColorDialog class	61	TControl class	84
Public constructors	61	Public constructors	84
Public member functions	61	Public member functions	84
Protected data members	61	Protected member functions	85
Protected member functions	62	Response table entries	86
Response table entries	62	TControlBar class	86
TChooseColorDialog::TData struct	62	Public constructors	87
Public data members	63	Public member functions	87
TChooseFontDialog class	63	Protected member functions	87
Public constructors	64	TControlGadget class	87
Protected data members	64	Public constructors and destructor	88
Protected member functions	64	Protected data members	88
Response table entries	65	Protected member functions	88
TChooseFontDialog::TData struct	65	Response table entries	89
Public data members	65	TCreatedDC class	89
TClientDC class	67	Public constructors and destructor	89
Public constructors	67	Protected member functions	90
TClipboard class	67	TCursor class	90
Public data members	67	Public constructors and destructor	90
Public member functions	67	Public member functions	91
Protected data members	70	TDC class	91
Protected constructors and destructor	70	Public data members	92

Public constructors and destructor	92	Protected data members	168
Public member functions	93	Protected member functions	168
Protected data members	134	TDocument::List class	169
Protected constructors	135	Public constructors and destructor	169
Protected member functions	136	Public member functions	169
TDecoratedFrame class	136	TDropInfo class	169
Public constructors	136	Public constructors	169
Public member functions	136	Public member functions	170
Protected data members	137	TEdgeConstraint struct	171
Protected member functions	137	Public member functions	171
Response table entries	138	TEdgeOrSizeConstraint struct	172
TDecoratedMDIFrame class	138	Public member functions	172
Public constructors	138	TEdit class	173
Protected member function	138	Public constructors	174
Response table entries	139	Public member functions	174
TDesktopDC class	139	Protected data members	179
Public constructors	139	Protected member functions	179
TDialog class	139	Response table entries	181
Public data members	140	TEditFile class	182
Public constructors and destructor	140	Public data members	182
Public member functions	140	Public constructors and destructor	182
Protected member functions	143	Public member functions	183
Response table entries	144	Protected member functions	184
TDialogAttr struct	144	Response table entries	185
Public data members	144	TEditSearch class	185
TDib class	145	Public data members	185
Protected data members	145	Public constructors	185
Public constructors and destructor	146	Public member functions	186
Public member functions	147	Response table entries	186
Protected member functions	151	TEditView class	187
TDibDC class	151	Public constructors and destructor	187
Public constructors	151	Public member functions	187
TDocManager class	152	Protected data member	188
Public data members	153	Protected member functions	188
Public constructors and destructor	153	Response table entries	189
Public member functions	153	TEventHandler class	189
Protected member functions	156	Public member functions	189
Response table entries	157	Protected member functions	190
TDocTemplate class	157	TEventHandler::TEqualOperator type	190
Public member functions	157	TEventHandler::TEventInfo class	190
Protected constructors and destructor	160	Public data members	190
Protected member functions	160	Public constructors	191
TDocTemplate<D,V> class	160	TEventStatus enum	191
Public constructors	161	TEventHandler::TEqualOperator type	191
Public member functions	161	TFileDocument class	192
TDocument class	162	Public constructors and destructor	192
Public data members	163	Public member functions	192
Public constructors and destructor	163	Protected data members	194
Public member functions	164	Protected member functions	194

TFileOpenDialog class	195	TGadgetWindowFont class	221
Public constructors	195	Public constructors	221
Public member functions	195	TGauge class	222
TFileSaveDialog class	195	Public member functions	222
Public constructors	195	Public constructors	223
Public member functions	196	Protected data members	223
TFilterValidator class	196	Protected member functions	223
Public constructors	196	Response table entries	224
Public member functions	196	TGdiObject class	224
Protected data members	197	Public data members	225
TFindDialog class	197	Public member functions	225
Public constructors	197	Protected data members	227
Protected member functions	197	Protected constructors and destructor	227
TFindReplaceDialog class	197	Macros	228
Public constructors	198	TGdiObject::TXGdi class	229
Public member functions	198	Public constructors	230
Protected data members	198	Public member functions	230
Protected member functions	199	TGroupBox class	230
Response table entries	199	Public data members	231
TFindReplaceDialog::TData struct	199	Public constructors	231
Public member functions	199	Public member functions	231
TFloatingFrame class	200	Protected member functions	232
Public constructors	201	THintMode enum	232
Public member functions	201	THSlider class	232
Response table entries	202	Public constructors	232
TFont class	202	Protected member functions	232
Public data members	202	TIC class	233
Protected data members	202	Public constructors	233
Public constructors	202	TIcon class	233
Public member functions	203	Public constructors and destructor	234
TFrameWindow class	203	Public member functions	235
Public data member	204	TInputDialog class	235
Public constructors and destructor	204	Public data members	235
Public member functions	204	Public constructors	236
Protected data members	206	Public member function	236
Protected member functions	206	Protected member function	236
Response table entries	208	TInStream class	236
TGadget class	208	Public constructors	236
Public data members	208	TKeyboardModeTracker class	237
Public constructors and destructor	209	Public data members	237
Public member functions	210	Public constructors	237
Protected data members	212	Protected data members	237
Protected member functions	213	Protected member functions	238
TGadgetWindow class	214	Response table entries	239
Public constructors and destructor	215	TLayoutConstraint struct	239
Public member functions	215	Public data members	240
Protected data members	218	TLayoutMetrics class	241
Protected member functions	219	Public data members	242
Response table entries	221	Public constructors	242

TLayoutWindow class	244	Public member functions	271
Examples	244	Protected data members	273
Public constructors and destructor	247	TMenuDescr class	273
Public member functions	247	Public data members	275
Protected data members	248	Public Constructors	275
Protected member functions	248	TMenuDescr::TGroup enum	276
Response table entries	248	TMessageBar class	276
TListBox class	248	Public constructors	276
Public constructors	248	Public member function	276
Public member functions	249	Protected data member	277
Protected member functions	254	Protected member functions	277
TListBoxData struct	254	TMetaFileDC class	277
Public data members	254	Constructors and destructor	277
Public constructors and destructor	255	Public member function	278
Public member functions	255	TMetaFilePict class	278
TListView class	256	Protected data member	278
Public constructors and destructor	256	Public constructors and destructor	278
Public data member	256	Public member functions	280
Public member functions	256	TModule class	280
Protected data members	257	Public data members	280
Protected member functions	257	Public constructors and destructor	281
Response table entry	259	Public member functions	282
TLookupValidator class	260	Protected data members	287
Public constructors	260	TModule::TXInvalidModule class	287
Public member functions	260	Public constructors	287
TMDIChild class	261	TOpenSaveDialog class	287
Public constructors and destructor	261	Public constructors	288
Public member functions	261	Public member functions	288
Protected member functions	262	Protected data members	288
Response table entries	263	Protected constructors	288
TMDIClient class	263	Protected member functions	289
Public data member	263	Response table entries	290
Public constructors and destructor	263	TOpenSaveDialog::TData struct	290
Public member functions	263	Public data members	290
Protected member functions	265	Public constructors and destructor	291
Response table entries	266	Public member functions	292
TMDIFrame class	267	TOutputStream class	292
Public data members	267	Public constructors	292
Public constructors	267	TPaintDC class	292
Public member functions	268	Public data members	292
Protected member function	268	Public constructors and destructor	292
Response table entries	268	TPalette class	293
TMeasurementUnits enum	269	Public data members	293
TMemoryDC class	269	Protected data members	293
Public constructors	269	Public constructors	293
Public member functions	269	Public member functions	295
Protected data member	270	Protected member functions	297
TMenu class	270	TPaletteEntry class	297
Public constructors and destructor	270	Public constructors	298

TPen class	298	TPrintout class	330
Public data members	298	Public constructors and destructor	330
Protected data members	299	Public member functions	330
Public constructors	299	Protected data members	332
Public member functions	300	TPrintoutFlags enum	332
TPicResult enum	300	TProcInstance class	333
TPlacement enum	301	Public constructors and destructor	333
TPoint class	301	Public member functions	333
Public constructors	301	TPXPictureValidator class	333
Public member functions	302	Public constructors	333
Friend functions	303	Public member functions	334
TPointer<> class	304	Protected data member	335
Public constructors	304	Protected member functions	335
Public member functions	304	TRadioButton class	336
TPopupMenu class	305	Public constructors	336
Public constructors	305	Protected member functions	337
Public member functions	305	Response table entries	337
TPreviewPage class	306	TRangeValidator class	337
Public constructors	307	Public constructors	337
Public member functions	307	Public member functions	337
Protected data members	307	Protected data members	338
Protected member functions	308	TRect class	338
Response table entries	308	Public constructors	339
TPrintDC class	308	Public member functions	340
Public constructors	308	Friend functions	344
Public member functions	308	TRegion class	345
Protected data members	319	Public data members	345
TPrintDialog class	319	Public constructors	345
Public constructors	319	Public member functions	346
Public member functions	319	TRelationship enum	349
Protected data members	320	TReplaceDialog class	349
Protected member functions	320	Public constructors	349
Response table entries	320	Protected member functions	349
TPrintDialog::TData struct	320	TResId class	350
Public data members	321	Public constructors	350
Public member functions	322	Public member functions	350
TPrintPreviewDC class	323	Friend functions	350
Public constructors and destructor	324	TResponseTableEntry class	351
Public member functions	324	Public data members	351
Protected data members	327	TRgbQuad class	352
Protected member functions	327	Public constructors	352
TPrinter class	327	TRgbTriple class	353
Public constructors and destructor	327	Public constructors	353
Public member functions	328	TScreenDC class	354
Protected data members	328	Public constructors	354
Protected member functions	329	TScrollBar	354
TPrinterAbortDlg class	329	Public data members	354
Public constructors	329	Public constructors	355
Public member functions	330	Public member functions	355

Protected member functions	357	Public member functions	381
TScrollBarData struct	358	Protected data members	382
Public data members	358	Protected member functions	382
TScroller class	358	TTileDirection enum	383
Public data members	359	TTinyCaption class	383
Public constructors and destructor	359	Protected data members	384
Public member functions	360	Protected constructors and destructor	385
TSeparatorGadget class	362	Protected member functions	385
Public member functions	362	Response table entries	389
TSize class	363	TToolBox class	389
Public constructors	363	Public constructors	390
Public member functions	364	Public member functions	390
Friend functions	365	Protected data members	390
TSlider class	365	Protected member functions	391
Public constructors and destructor	367	TTransferDirection enum	391
Public member functions	367	TValidator class	391
Protected member functions	368	Public constructors and destructor	392
Protected data members	371	Public member functions	392
Response table entries	372	Protected data members	394
TSortedStringArray typedef	373	TValidator::TXValidator class	395
TStatic class	373	Public constructors	395
Public data members	373	TVbxControl class	395
Public constructors	373	Public constructors and destructor	396
Public member functions	374	Public member functions	397
Protected member functions	375	Protected member functions	401
TStatus class	375	Response table entries	402
Public constructors	375	TVbxEventHandler class	402
Public data members	375	Protected member functions	407
TStatusBar class	375	Response table entries	407
Public data members	376	TView class	407
Public constructors	376	Public data members	408
Public member functions	376	Public constructors and destructor	408
Protected data members	377	Public member functions	409
Protected member functions	378	Protected data members	410
TStream class	378	Protected member functions	410
Public data members	378	TVSlider class	411
Public destructor	378	Public constructors	411
Public member functions	379	Protected member functions	411
Protected data members	379	TWidthHeight enum	412
Protected constructors	379	TWindow class	412
TStringLookupValidator class	379	Public data members	413
Public constructors and destructor	379	Public constructors and destructor	414
Public member functions	380	Public member functions	415
Protected data member	380	Protected data members	445
TSystemMenu class	380	Protected member functions	446
Public constructors	380	Response table entries	447
TTextGadget class	381	TWindow::TXWindow class	448
Public data members	381	Public constructors	448
Public constructors	381	Public data members	448

Public member functions	448	Chapter 4 WIN API encapsulated functions	473
TWindowFlag enum	449	Appendix A Inheritance diagrams	479
TWindowAttr struct	449	TApplication	479
Public data members	449	TBitMap	480
TWindowDC class	450	TBitMapGadget	480
Public constructors and destructor	450	TButton	481
Protected constructors	451	TButtonGadget	482
Protected data member	451	TCheckbox	483
TWindowView class	451	TChooseColorDialog	484
Public constructors and destructor	451	TChooseFontDialog	486
Public member functions	451	TClipboardViewer	487
Response table entries	452	TComboBox	488
TXCompatibility class	452	TComboBoxData	490
Public constructors	453	TCommonDialog	492
Public member functions	453	TControl	493
TXOwl class	453	TControlBar	494
Public constructors and destructor	454	TControlGadget	495
Public member functions	454	TDecoratedFrame	496
Vnxxxx view notification constants	455	TDecoratedMDIFrame	497
Voxxxx validator constants	455	TDialog	499
xs exception status enum	456	TDocManager	500
Chapter 2 Event handlers	457	TDocTemplate	500
Chapter 3 Dispatch functions	465	TDocument	500
HBRUSH_HDC_W_U_Dispatch	466	TEdit	501
i_LPARAM_Dispatch	466	TEditFile	503
i_U_W_U_Dispatch	466	TEditSearch	505
i_WPARAM_Dispatch	467	TEditView	506
LRESULT_WPARAM_LPARAM_Dispatch	467	TFileDocument	507
U_POINT_Dispatch	467	TFileOpenDialog	508
U_U_U_U_Dispatch	467	TFileSaveDialog	510
U_U_U_W_Dispatch	468	TFindDialog	512
U_Dispatch	468	TFloatingFrame	514
U_WPARAM_LPARAM_Dispatch	468	TGadgetWindow	516
v_LPARAM_Dispatch	468	TGauge	518
v_POINT_Dispatch	468	TLayoutWindow	519
v_POINTER_Dispatch	469	TListBox	520
v_U_B_W_Dispatch	469	TListView	521
v_U_POINT_Dispatch	469	TMDIClient	522
v_U_U_Dispatch	469	TMDIFrame	523
v_U_U_U_Dispatch	470	TRadioButton	525
v_U_U_W_Dispatch	470	TSlider	527
v_Dispatch	470	TStatusBar	529
v_WPARAM_Dispatch	470	TTextGadget	530
v_WPARAM_LPARAM_Dispatch	471	TToolBox	531
v_W_W_Dispatch	471	TWindowView	533
		TWindow	534
		Index	535

Tables

1.1 Summary of static and import libraries	10	1.33 Shift key bit values	406
1.2 Summary of dynamic link libraries	11	1.34 Mouse button key arguments	406
1.3 Summary of header files	11	1.35 TWindow's attribute masks	449
1.4 Summary of resource files	15	1.36 Vnxxxx view notification IDs	455
1.5 Button flag constants	16	1.37 Voxxxx validator constants	455
1.6 Command-based constants	16	1.38 xs exception status enum	456
1.7 Command-based constants	17	2.1 WM_COMMAND messages	458
1.8 Command-based constant	17	2.2 WM_XXXX Window messages	458
1.9 Command-based constants	17	2.3 Child ID notification messages	461
1.10 Command message constants	18	2.4 Button notification messages	461
1.11 Document manager mode constants	19	2.5 Combo box notification messages	461
1.12 Document message enum	20	2.6 List box notification messages	462
1.13 Document view constants	20	2.7 Edit control notification messages	462
1.14 EV_XXXX macros	22	2.8 New document and view messages	462
1.15 ID file constants	22	2.9 Document view messages	463
1.16 ID printer constants	23	2.10 VBX messages	463
1.17 Accelerator ID constants	23	4.1 Encapsulated inline HWND functions	474
1.18 Menu ID constants	23	4.2 Encapsulated Window messages	474
1.19 Document string ID constants	23	4.3 Window coordinates and dimensions	474
1.20 Edit file ID constants	24	4.4 Window properties	475
1.21 Exception message constants	24	4.5 Window placement	475
1.22 Listview string ID constants	25	4.6 Window relationships	476
1.23 Printer string ID constants	25	4.7 Window painting functions	476
1.24 Validator ID constants	25	4.8 Window scrolling functions	476
1.25 shxxxx constants	30	4.9 Child window ID functions	477
1.26 TCheckBox check states	59	4.10 Menu and menu bar functions	477
1.27 Event status constants	191	4.11 Clipboard functions	477
1.28 Picture format characters	335	4.12 Timer functions	478
1.29 Transfer function constants	391	4.13 Caret and cursor functions	478
1.30 Property and C++ types	399	4.14 Hot key functions	478
1.31 Basic and C++ VBX data types	405	4.15 Help and task functions	478
1.32 VBX event arguments	405		

Introduction

This *Reference Guide* can be used to help you perform the following tasks in ObjectWindows:

- Look up the overall purpose for each class.
- Learn the details about how to use a particular ObjectWindows class and its members and functions.
- View the virtual and nonvirtual multiple inheritance relationships among ObjectWindows classes.
- Learn which classes introduce or redefine functions.
- Determine which ancestor of a class introduced a data member or member function.
- Learn how data members and member functions are declared.
- Use event-handling functions to respond to messages.
- Use dispatch functions to crack Windows messages.

Contents of this manual

This manual has four reference chapters and one appendix:

Chapter 1: Library reference is an alphabetical listing of all the standard ObjectWindows classes, including explanations of their purpose, usage, and members. It also describes the nonobject elements such as structures, constants, variables, and macros that classes use.

Chapter 2: Event handlers lists the ObjectWindows functions and notification codes that crack Windows messages.

Chapter 3: Dispatch functions lists all of the ObjectWindows functions that dispatch Windows messages.

Chapter 4: WIN API encapsulated functions lists the ObjectWindows functions that encapsulate Windows API functions.

Appendix A: Inheritance diagrams lists the member functions for each class and shows which functions override functions defined in the base class.

Icons and typefaces used in this manual



Depending on the Windows application programming (API) environment you are using, different ObjectWindows functions are available. If a data member or function is available only under the Windows 32-bit API, including Windows NT, the Win32 icon is displayed to the left of the data member or function.



Similarly, if a class member is functional only under Win16, the Win16 icon is displayed to the left of the data member or function. Any differences in implementation between Win16 and Win32 are described. Otherwise, a class member is considered fully functional under both the Win16 and Win32 APIs.

Boldface Boldface type indicates language keywords (such as **char**, **switch**, and **begin**) and command-line options (such as **-rn**).

Italics Italic type indicates program variables and constants that appear in text. This typeface is also used to emphasize certain words, such as new terms.

Monospace Monospace type represents text as it appears onscreen or in a program. It is also used for anything you must type literally (such as `TD32` to start up the 32-bit Turbo Debugger).

Key1 This typeface indicates a key on your keyboard. For example, "Press *Esc* to exit a menu."

Key1+Key2 Key combinations produced by holding down one or more keys simultaneously are represented as *Key1+Key2*. For example, you can execute the Program Reset command by holding down the *Ctrl* key and pressing *F2* (which is represented as *Ctrl+F2*).

MenuCommand This command sequence represents a choice from the menu bar followed by a menu choice. For example, the command "File | Open" represents the Open command on the File menu.



This icon indicates material you should take special notice of.

Conventions used in this manual

Inline functions, those functions that are declared and defined within a class, are prefaced by the keyword “inline” before the function declaration. For example,

```
inline virtual int AddString(const char far* string);
```

Cross-referenced entries to ObjectWindows functions include the class name, the scope resolution operator, and the function name. For example,

See also: *TApplication::PumpWaitingMessages*

Windows API function calls are prefixed with the scope resolution operator (::). For example,

See also: *::ShowWindow*

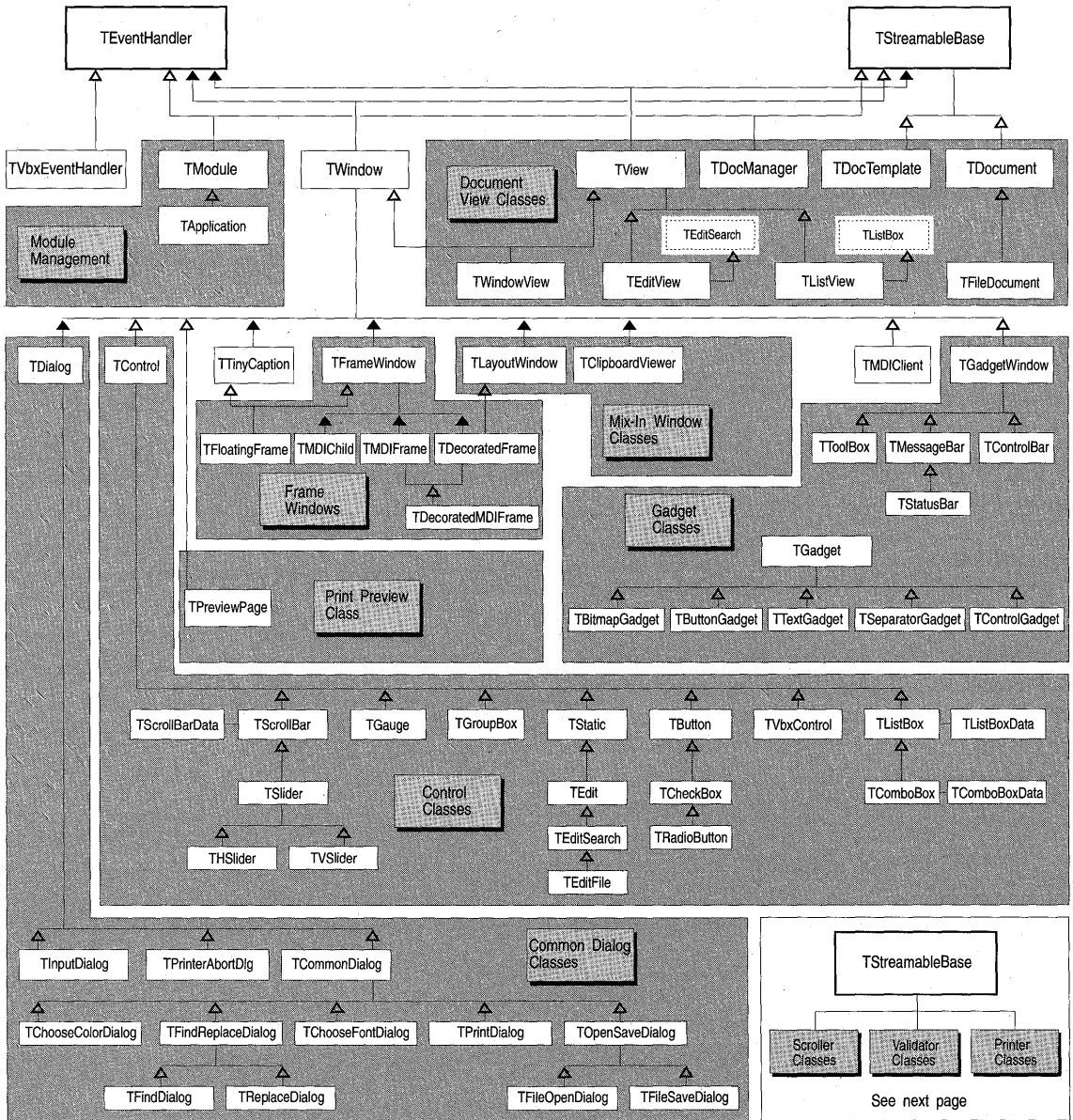
C++ data types that are keywords (such as **int** and **long**) are in lowercase bold. Predefined Windows types (such as **HWND** and **UINT**) are in capital letters; for example,

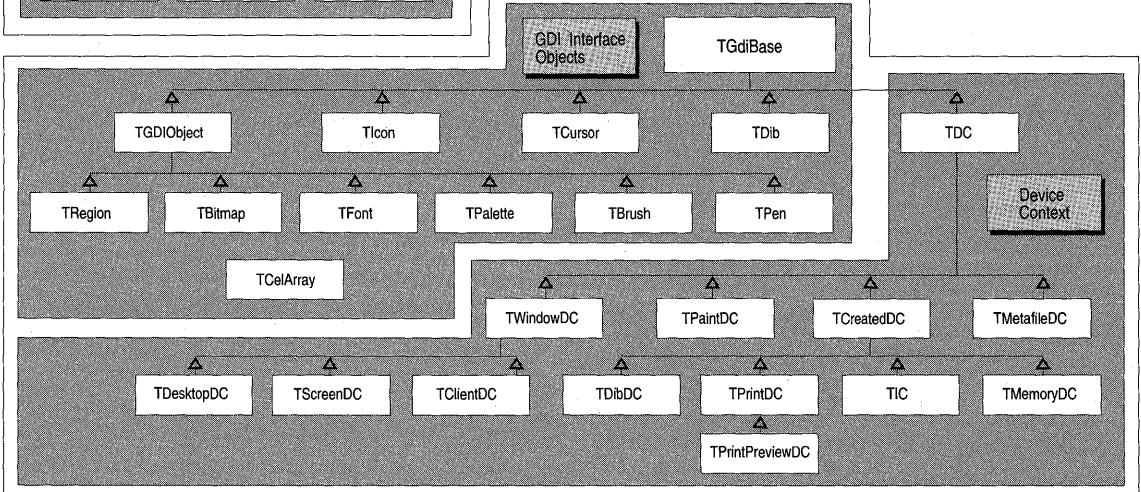
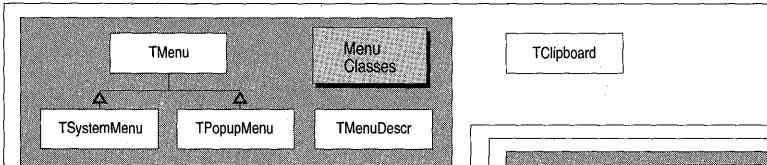
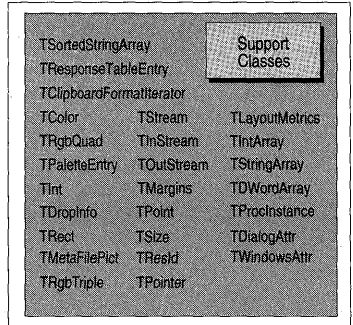
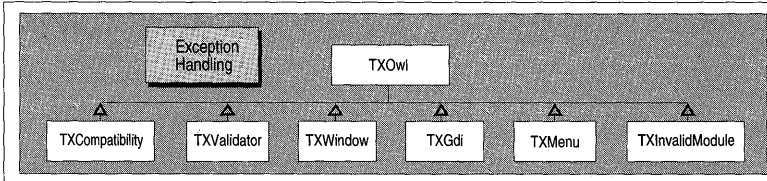
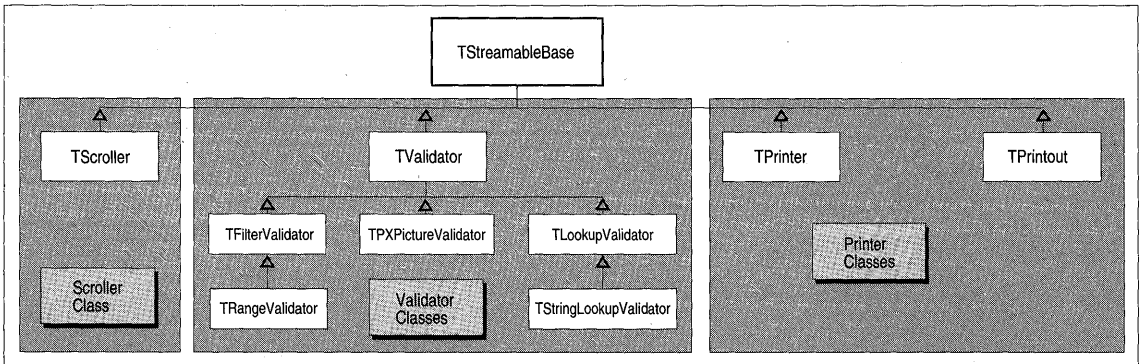
```
inline BOOL TrackPopupMenu(UINT flags, int x, int y, int rsvd, HWND wnd,  
                          TRect* rect=0);
```

ObjectWindows hierarchy diagram

The ObjectWindows hierarchy diagram shows the classes that are described in this manual. The classes are grouped according to functional categories, and all related classes are in one shaded unit. A class is enclosed in dashed lines if it is a parent class for a multiply-inherited class. For example, *TListBox* is the parent class for *TListView*, which is derived from both *TView* and *TListBox*. The second page shows additional classes.

ObjectWindows class hierarchy





△ Nonvirtual inheritance ▲ Virtual inheritance

Library reference

The ObjectWindows library-reference entries begin on page 16. A sample entry that explains the contents of each entry section is provided on page 7.

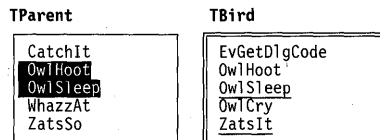
For more information on Windows types such as LPSTR, see the online Help.

This chapter alphabetically lists the ObjectWindows classes, data members, member functions, macros, constants, and data types. The header file that defines each entry is listed opposite the entry name. Class members are grouped according to their access specifiers—public or protected. Within these categories, data members, then constructors (and the destructor, if one exists), and member functions are listed alphabetically.

Because many of the properties of the classes in the hierarchy are inherited from base classes, only data members and member functions that are *new* or *redefined* for a particular class are listed. Private members are not listed. If any response table entries exist, they are also listed. The cross-referenced entries provide additional information about how to use the specified entry. The first sample entry (on page 7) illustrates this format.

To find information about a particular inherited member function, use the inheritance diagram in A. The inheritance diagram shows the ancestry of the class, excluding *TEventHandler* and *TStreamableBase*, from which all classes are inherited.

The following figure uses a sample class and its functions to illustrate this format. In this example, *TBird* inherits functions from *TParent* and overrides the shaded member functions, *OwlHoot* and *OwlSleep*, defined in the base class. The underlined functions *OwlSleep* and *ZatsIt* are defined as virtual functions in *TBird*. For more information about virtual functions (those functions defined in base classes and overridden in derived classes), see the *ObjectWindows Programmer's Guide*.



TBird class [sample]**Public data members**

This section alphabetically lists all public data members and their declarations, and explains how they are used.

anOwlBeak

```
anOwlType anOwlBeak;
```

anOwlBeak is a data member that holds information about this sample class. This text explains what *anOwlBeak* contains, and how you use it.

See also: Related data members, member functions, classes, constants, and types

anOwlWing

```
anOwlType anOwlWing;
```

anOwlWing is another public data member.

Public constructors and destructor

This section lists any public constructors and destructor for this class. Classes can have more than one constructor; they never have more than one destructor.

Constructor

```
TBird(anOwlType aParameter);
```

Constructor for a new sample class; sets the *anOwlBeak* data member to *aParameter*.

Destructor

```
~TBird;
```

Destructor for a new sample class; destroys the *TBird* object.

Public member functions

This section alphabetically lists all public member functions that are either newly defined for this class or that are redefined inherited member functions. If a function overrides a virtual base class function, the text specifies this.

EvGetDlgCode

```
UINT OwlHoot();
```

Responds to WM_GETDLGCODE messages.

OwlHoot

```
void OwlHoot();
```

The *OwlHoot* member function causes the sample class to perform some action. This function overrides the function *OwlHoot* in its base class, *TParent*.

See also: *TParent::OwlHoot*

OwlSleep

```
virtual int OwlSleep(int index);
```

The *OwlSleep* function performs another action and overrides the function *OwlSleep* in its base class, *TParent*.

See also: *TParent::OwlSleep*

Protected data members

This section alphabetically lists all protected data members and their declarations, and explains how they are used.

anOwlFeather

```
anOwlType anOwlFeather;
```

anOwlFeather is a protected data member that holds information about this sample class.

See also: Related data members, member functions, classes, constants, and types

Protected constructors

Constructor

```
TBird(anOwlType bParameter);
```

If the class has a protected constructor, it is listed here.

Protected member functions

This section lists all protected member functions.

OwlCry

```
BOOLEAN OwlCry;
```

The *OwlCry* member function causes the sample class to perform some action.

See also: *TSomethingElse::OwlCry*

ZatsIt

```
virtual int ZatsIt(int index);
```

The *ZatsIt* function performs a particular function in class *TBird*.

Response table entries

The *TBird* response table contains this predefined macro for the EV_XXXX messages and calls this member function:

Response table entry	Member function
EV_WM_GETDLGCODE	EVGetDlgCode

ObjectWindows libraries

The following table lists the ObjectWindows static libraries, their uses, and the operating system under which the library is available. These files are in your library directory.

The name of the OWLWx.LIB file varies, depending on several factors—whether you are building a small, medium, or large memory model application or a WIN16 or WIN32 application. For example, if the application is built for a 16-bit, small memory model, the name of the library file is OWLWS.LIB. If you're building a flat model WIN32 application, the name of the library file is OWLWF.LIB where "F" indicates a flat model application.

Basic versions of the ObjectWindows files are included on your installation disk. You can build the additional versions by invoking the ObjectWindows makefile located in your SOURCE\OWL subdirectory using the -DDIAGS and -DMODEL switches.

Table 1.1
Summary of static
and import libraries

File name	Application	Use
OWLWS.LIB	Win16	16-bit small model
OWLWM.LIB	Win16	16-bit medium model
OWLWL.LIB	Win16	16-bit large model
OWLDWS.LIB	Win16	16-bit diagnostic small model
OWLDWM.LIB	Win16	16-bit diagnostic medium model
OWLDWL.LIB	Win16	16-bit diagnostic large model
OWLWF.LIB	Win32s, Win32	32-bit library
OWLDWF.LIB	Win32, Win32s	32-bit diagnostic library
OWLWI.LIB	Win16	16-bit import library for OWL200.DLL
OWLDWI.LIB	Win16	16-bit import library for OWL200D.DLL
OWLWFI.LIB	Win32, Win32s	32-bit import library for OWL200F.DLL
OWLDWFI.LIB	Win32, Win32s	32-bit import library for OWL200DF.DLL

The dynamic-link (DLL) versions of ObjectWindows are contained in your \BIN subdirectory. The following table lists the DLL names and uses.

Table 1.2
Summary of dynamic
link libraries

File name	Application	Use
OWL200.DLL	Win 16	16-bit dynamic library
OWL200F.DLL	Win 32	32-bit dynamic library
OWL200D.DLL	Win 16	Diagnostic version of 16-bit dynamic library
OWL200DF.DLL	Win 32	Diagnostic version of 32-bit dynamic library

The ObjectWindows header files

Header files, located in your OWL\INCLUDE subdirectory, contain declarations for class functions and definitions for data types and constants.

Table 1.3: Summary of header files

File name	Class definition	Use
applicat.h	TApplication	Controls the basic behavior of all ObjectWindows applications.
bitmapga.h	TBitMapGadget	Displays an array of bitmap images.
bitset.h	TBitSet TCharSet	Sets or clears one or more bits. Sets or clears bytes.
button.h	TButton	Creates different types of button controls.
buttonga.h	TButtonGadget	Creates button gadgets that can be clicked on or off.
celarray.h	TCelArray	Creates an array of cels.
checkbox.h	TCheckBox	Represents a check box control.
chooseco.h	TChooseColor	Represents modal dialog boxes that allow color selection.
choosefo.h	TChooseFont	Represents modal dialog boxes that allow font selection.
clipboar.h	TClipboard	Contains functions that control how Clipboard data is handled.
clipview.h	TClipboardViewer	Registers a TClipboardViewer as a Clipboard viewer.
color.h	TColor	Contains functions used to simplify standard Windows color operations.

Table 1.3: Summary of header files (continued)

combobox.h	TCombobox	Creates combo boxes or combo box controls in a window, and class TComboBoxData, which is used to transfer data between combo boxes.
commdial.h	TCommonDialog	Abstract base class for TCommonDialog objects.
compat.h		Defines functions and constants used internally by ObjectWindows.
control.h	TControl	Used to create control objects in derived classes.
controlb.h	TControlBar	Implements a control bar that provides mnemonic access for its button gadgets.
controlg.h	TControlGadget	Allows controls to be placed in a gadget window.
dc.h	TBandInfo, TClientDC, TCreatedDC, TDC, TDesktopDC, TDibDC TIC, TMemoryDC, TMetaFileDC, TPaintDC TPrintDC, TScreenDC, TWindowDC	GDI DC wrapper classes that create DC objects.
deccframe.h	TDecoratedFrame	Creates a client window into which decorations can be placed.
decmdifr.h	TDecoratedMDIFrame	Creates a frame object that supports decorated child windows.
dialog.h	TDialog TDialogAttr	Creates modal and modeless dialog box interface elements. Holds the dialog box element's attributes.
dispatch.h		Defines dispatch functions designed to crack Windows messages.
docmanag.h	TDocManager TDocTemplate	Creates a document manager object that manages the documents and templates. Creates the templates.
docview.h	TDocument, TView, TWindowView, TStream, TInStream, TOutStream	Create, destroy, and send messages about document views.
edit.h	TEdit	Creates an edit control interface element.
editfile.h	TEditFile	Creates a file editing window.
editsear.h	TEditSearch	Creates an edit control that responds to search and replace commands.

Table 1.3: Summary of header files (continued)

editview.h	TEditView	View wrapper for TEdit.
eventhan.h	TEventHandler	Used to derive class capable of handling messages.
except.h	TXOwl TXCompatibility TXOutOfMemory TStatus	Base exception-handling class. Describes a status exception. Describes an out-of-memory exception. Included for backward compatibility.
filedoc.h	TFileDocument	Opens and closes document views.
findrepl.h	TFindDialog, TFindReplaceDialog::	These classes create and define the attributes of modeless dialog boxes that respond to search and replace commands.
floatfra.h	TFloatingFrame	Implements a floating frame within a parent window.
framewin.h	TFrameWindow TMenuDescr	Controls window-specific behavior such as keyboard navigation and command processing. Describes a menu bar.
gadget.h	TGadget	Creates gadget objects that belong to a gadget window and have specified attributes.
gadgetwi.h	TGadgetWindow	Maintains a list of tiled gadgets for a window.
gauge.h	TGauge	Establishes the behavior of gauge controls.
gdibase.h	TGdiBase	Abstract base class for all GDI classes.
gdiobjec.h	TGdiObject TPen, TBrush, TFont, TPalette, TBitmap, TIcon, TCursor, TDib, TRegion.	Base GDI class. These classes create specified GDI objects.
groupbox.h	TGroupBox	Creates a group box object that represents a group box element in Windows.
inputdia.h	TInputDialog	Generic dialog box.
keymodet.h	TKeyboardModeTracker	A mix-in class designed to track changes in keyboard modes.
layoutco.h	TLayoutConstraint	Creates layout constraints.
layoutwi.h	TLayoutMetrics	Contains the layout constraints used to define the layout metrics for a window.
listbox.h	TListBox TListBoxData	Creates a list box object. Used to transfer the contents of a list box.
listview.h	TListView	Provides views for list boxes.

Table 1.3: Summary of header files (continued)

mdi.h	TMDIClient TMDIFrame	Manages MDI child windows. The main window of MDI-compliant applications.
mdichild.h	TMDIChild	Defines the behavior of MDI child windows.
menu.h	TMenu, TPopupMenu, TSystemMenu	Create menu objects.
messageb.h	TMessageBar	Implements a message bar.
metafile.h	TMetaFilePict	A wrapper class used with TMetaFileDC.
module.h	TModule	Defines the basic behavior for ObjectWindows libraries and applications.
opensave.h	TOpenSave	Base class for modal open and save dialog boxes.
owlall.h		Include file for all of the ObjectWindows classes.
owlcore.h		Include file for the core ObjectWindows classes.
owldefs.h		Includes definitions of macros used by all ObjectWindows programs.
owlpch.h		Contains definitions of macros, data, and functions used by ObjectWindows.
point.h	TPoint, TSize, TRect TDropInfo TProclInstance TPointer	Mathematical classes. Supports file-name drag and drop operations. A Win16 support class. Provides exception-safe pointer manipulation.
preview.h	TPreviewPage	Displays a document page in a print preview window.
printdia.h	TPrintDialog	Displays a modal print or print setup dialog box.
printer.h	TPrinter TPrintout TPrinterAbortDlg	Represents the printer device. Represents the printed document. Represents the printer-abort dialog box.
radiobut.h	TRadioButton	Create a radio button control.
scrollba.h	TScrollBar TScrollBarData	Represents a vertical or horizontal scroll bar control. Contains the values of the thumb position on the scroll bar.
scroller.h	TScroller	Implements automatic window scrolling.
signatur.h		Defines the message cracking signature templates used by ObjectWindows event-handling functions.

Table 1.3: Summary of header files (continued)

slider.h	TSlider	Defines the basic behavior of sliders.
	THSlider	A horizontal slider.
	TVSlider	A vertical slider.
static.h	TStatic	Create a static control in a window.
statusba.h	TStatusBar	Constructs a status bar.
textgadg.h	TTextGadget	Construct a text gadget object.
tinycapt.h	TTinyCaption	Produces a smaller caption bar for a window.
toolbox.h	TToolBox	Creates a toolbox object with a specified number of rows and columns.
	TValidator	Base validator class.
	TPXPictureValidator	Picture validator.
	TFilterValidator	Filter validator.
	TRangeValidator	Range validator.
	TLookupValidator	Lookup validation.
vbctl.h	TStringLookupValidator	String validation.
	TVbxControl	Interface for VBX controls.
version.h	TVbxEventHandler	Handles events from VBX controls.
		Defines the internal version number of the ObjectWindows library.
window.h	TWindow	Provides window-specific behavior and encapsulates many of the Windows API functions.
windowev.h		Defines event handlers and response table macros for Windows messages.

The ObjectWindows resource files

The ObjectWindows resource files define resource and command IDs.

Table 1.4
Summary of resource
files

File name	Use
Directory of OWLINCLUDE	
docview.rh	Defines resource and command IDs to use with docview.h and docview.rc.
edit.rh	Defines command IDs to use with edit.h.
editfile.rh	Defines resource and command IDs to with in editfile.rc and editfile.h.
editsear.rh	Defines resource and command IDs to use in editsear.rc and editsear.h.
except.rh	Defines string resource IDs to use with except.h and except.rc.

Table 1.4: Summary of resource files (continued)

inputdia.rh	Defines resource IDs to use with inputdia.rc and inputdia.h.
mdi.rh	Defines resource & command IDs to use with mdi.h.
printer.rh	Defines resource IDs to use with printer.rc and printer.h.
slider.rh	Defines resource IDs to use with slider.h.
validate.rh	Defines resources to use with TValidator and derived classes.
window.rh	Defines command IDs to use with window.h.

The ObjectWindows library reference

The following section lists the classes, data types, and symbolic constants used by ObjectWindows applications.

BF_xxxx constants

checkbox.h

Check box and radio button objects use the button flag constants to indicate the state of a selection box.

Table 1.5
Button flag constants

Constant	Meaning
BF_CHECKED	Item is checked.
BF_GRAYED	Item is grayed.
BF_UNCHECKED	Item is unchecked.

See also: *TCheckbox::GetCheck*, *TCheckbox::SetCheck*

CM_xxxx edit constants

edit.h

TEdit defines these command-based member functions, which are invoked in response to a particular edit menu selection or command.

Table 1.6
Command-based constants

Constant	Member function	Menu equivalent
CM_EDITCLEAR	<i>TEdit::CMEditClear</i>	Edit Clear
CM_EDITCOPY	<i>TEdit::CMEditCopy</i>	Edit Copy
CM_EDITCUT	<i>TEdit::CMEditCut</i>	Edit Cut
CM_EDITDELETE	<i>TEdit::CMEditDelete</i>	Edit Delete
CM_EDITPASTE	<i>TEdit::CMEditPaste</i>	Edit Paste
CM_EDITUNDO	<i>TEdit::CMEditUndo</i>	Edit Undo

CM_xxxx edit file constants**docview.rh**

These command-based member functions are invoked in response to open, close, print, and save commands.

Table 1.7
Command-based
constants

Constant	Member function	Menu equivalent
CM_FILECLOSE		
CM_FILENEW	<i>TEditFile::CmFileNew</i>	File New
CM_FILEOPEN	<i>TEditFile::CmFileOpen</i>	File Open
CM_FILEPRINT		File Print
CM_FILEPRINTERSETUP		File Printer Setup
CM_FILESAVE	<i>TEditFile::CmFileSave</i>	File Save
CM_FILESAVEAS	<i>TEditFile::CmFileSaveAs</i>	File Save As

CM_xxxx edit file exit constant**window.rh**

This command-based member function is invoked in response to a file exit request from a user.

Table 1.8
Command-based
constant

Constant	Member function	Menu equivalent
CM_EXIT	<i>TWindow::CmExit</i>	File Exit

CM_xxxx edit replace constants**editsear.rh**

These command-based member functions are invoked when the corresponding find and replace command is received.

Table 1.9
Command-based
constants

Constant	Member function	Menu equivalent
CM_EDITFIND	<i>TEditWindow::CMEditFind</i>	Edit Find
CM_EDITFINDNEXT	<i>TEditWindow::CMEditFindNext</i>	Edit Find Next
CM_EDITREPLACE	<i>TEditWindow::CMEditReplace</i>	Edit Replace

CM_xxxx MDI constants

mdi.rh

These MDI functions are invoked when the corresponding MDI command message is received.

Table 1.10
Command message
constants

Constant	Member function	Menu equivalent
CM_ARRANGEICONS	<i>TMDIClient::CmArrangeIcons</i>	Window Arrange Icons
CM_CASCADECHILDREN	<i>TMDIClient::CmCascadeChildren</i>	Window Cascade
CM_CLOSECHILDREN	<i>TMDIClient::CmCloseChildren</i>	Window Close All
CM_CREATECHILD	<i>TMDIClient::CmCreateChild</i>	
CM_TILECHILDREN	<i>TMDIClient::CmTileChildren</i>	Window Tile

DECLARE_RESPONSE_TABLE macro

eventhan.h

To handle events for a class, you need to declare a response table using the `DECLARE_RESPONSE_TABLE` macro within the class definition and you need to define the response table using one of the `DEFINE_RESPONSE_TABLE` macros. For example, to declare a response table, use

```
DECLARE_RESPONSE_TABLE(Class);
```

where *Class* represents the name of the current class.

See also: `DEFINE_RESPONSE_TABLE` macros, `END_RESPONSE_TABLE` macro

For more information about response table entries, see *TEventHandler* (where the response table entries are defined) or *TWindow* (where the member functions are defined). For more information about how to declare, define, and add message response entries to a response table, see Chapter 5 in the *Object Windows Programmer's Guide*.

DEFINE_DOC_TEMPLATE_CLASS macro

docmanag.h

Used to create a document template, the `DEFINE_DOC_TEMPLATE_CLASS` takes three arguments: the name of the document class that holds the data, the name of the view class that displays the data, and the name of the template class. The following examples from `DVSAMPLE.CPP`, a sample program on your distribution disk, associate document and view classes with new template classes.

```
DEFINE_DOC_TEMPLATE_CLASS(TFileDocument, TListView, ListTemplate);
DEFINE_DOC_TEMPLATE_CLASS(TFileDocument, TEditView, EditTemplate);
```

See also: *TDocTemplate*

DEFINE_RESPONSE_TABLE macros

eventhan.h

The `DEFINE_RESPONSE_TABLE x` macro takes one plus x number of arguments: the name of the class that is defining the response table, and its immediate as well as any virtual base classes. Use the `END_RESPONSE_TABLE` macro to end the definition for the response table. Between the `DEFINE_RESPONSE_TABLE` and `END_RESPONSE_TABLE` macros, you might need to insert the message response entries. For example,

```
DEFINE_RESPONSE_TABLE1(TMyClass, TWindow)
    EV_WM_PAINT,
    EV_WM_LBUTTONDOWN,
END_RESPONSE_TABLE;
```

In this example, `EV_WM_PAINT` and `EV_WM_LBUTTONDOWN` illustrate the message response entries for the class *TMyClass* derived from *TWindow*.

The following table shows the form the `DEFINE_RESPONSE_TABLE` macro takes depending on the number of base classes.

Base classes	Macro
0	<code>DEFINE_RESPONSE_TABLE(Class)</code>
1	<code>DEFINE_RESPONSE_TABLE(Class, Base)</code>
2	<code>DEFINE_RESPONSE_TABLE2(Class, Base1, Base2)</code>
3	<code>DEFINE_RESPONSE_TABLE3(Class, Base1, Base2, Base3)</code>

See also: `END_RESPONSE_TABLE` macro

dmxxxx document manager mode constants

docmanag.h

TDocManager uses the *dmxxxx* constants to indicate if a document supports single or multiple open documents, and to indicate if it has file menu IDs.

Table 1.11
Document manager
mode constants

Constant	Meaning
<code>dmMenu</code>	Sets IDs for file menu.
<code>dmMDI</code>	Supports multiple open documents.

Table 1.11: Document manager mode constants (continued)

dmNoRevert	Disables the FileRevert menu command.
dmSaveEnable	Enables FileSave menu command.
dmSDI	Does not support multiple open documents.

See also: *TDocManager::TDocManager*

dnxxxx document message enum

docmanag.h

TDocManager uses the *dnxxxx* message constants to indicate that a document or view has been created or closed. You can set up response table entries for these messages using the `EV_OWLVIEW` or `EV_OWLDOCUMENT` macros. See Chapter 9 in the *Object Windows Programmer's Guide* for information about how to do this.

Table 1.12
Document message
enum

Constant	Meaning
dnCreate	A new document or view has been created.
dnClose	A document or view has been closed.

See also: *TDocManager::TDocManager*

dt document view constants

docmanag.h

dtxxxx constants are used by *TDocument* and *TDocTemplate* to create templates. Several constants are equivalent to the `OFN_xxxx` constants defined by Windows in `commdlg.h`.

Table 1.13: Document view constants

Constant	Windows equivalent	Meaning
dtAutoDelete		Deletes the document when the last view is deleted.
dtAutoOpen		Opens a document upon creation.
dtCreatePrompt	(<code>OFN_CREATEPROMPT</code>)	Prompts the user before creating a document that does not currently exist.
dtFileMustExist	(<code>OFN_FILEMUSTEXIST</code>)	Lets the user enter only existing file names in the File Name entry field. If an invalid file name is entered, causes a warning message to be displayed.
dtHidden		Hides the template from the user's selection.
dtHideReadOnly	(<code>OFN_HIDEREADONLY</code>)	Hides the read-only check box.
dtNewDoc		Creates a new document with no path specified.

Table 1.13: Document view constants (continued)

dtNoAutoView		Does not automatically create the default view type.
dtNoReadOnly	(OFN_NOREADONLYRETURN)	Returns the specified file as writeable.
dtNoTestCreate	(OFN_NOTESTFILECREATE)	Does not perform document-creation tests. The file is created after the dialog box is closed. If the application sets this flag, there is no check against write protection, a full disk, an open drive door, or network protection. For certain network environments, this flag should be set.
dtOverwritePrompt	(OFN_OVERWRITEPROMPT)	When the Save As dialog box is displayed, asks the user if it's OK to overwrite the file.
dtPathMustExist	(OFN_PATHMUSTEXIST)	Allows only valid document paths to be entered. If an invalid path name is entered, causes a warning message to be displayed.
dtProhibited	(OFN_ALLOWMULTISELECT) (OFN_ENABLEHOOK) (OFN_ENABLETEMPLATE) (OFN_ENABLETEMPLATEHANDLE)	Doesn't support these specified Windows options.
dtReadOnly	(OFN_READONLY)	Checks the read-only check box when the dialog box is created.
dtSelected		Indicates the last selected template.
dtSingleView		Provides only a single view for each document.
dtUpdateDir		Updates the directory with the dialog directory.

See also: *TDocument::CreateFlags*, *TDocTemplate::Flags*

END_RESPONSE_TABLE macro

eventhan.h

This macro indicates the end of a response table.

```
END_RESPONSE_TABLE;
```

See also: `DEFINE_RESPONSE_TABLE` macro

EV_xxxx macros

windowev.h

The `EV_xxxx` macros create response table entries that match events to member functions. See Chapter 2 for a list of the `EV_xxxx` macros and their corresponding functions and Chapter 5 in *Object Windows Programmer's Guide* for details about how to use these macros.

Table 1.14: EV_xxxx macros

Macro	Meaning
EV_CHILD_NOTIFY(id, notifyCode, method)	Handles child ID notifications (for example, button, edit control, list box, combo box, and scroll bar notification messages) at the child's parent window. Passes no arguments.
EV_CHILD_NOTIFY_ALL_CODES	Passes all notifications to the response function and passes the notification code in as an argument.
EV_CHILD_NOTIFY_AND_CODE(id, notifyCode, method)	Handles child ID notifications at the child's parent window and passes the notification code as an argument.
EV_COMMAND(id, method)	Handler for menu selections, accelerator keys, and push buttons.
EV_COMMAND_AND_ID(id, method)	Handler for multiple commands using a single response function. Passes the menu ID in as an argument.
EV_COMMAND_ENABLE(id, method)	Enables and disables commands such as buttons and menu items.
EV_MESSAGE(message, method)	General purpose macro for Windows WM_xxxx messages.
EV_NOTIFY_AT_CHILD(notifyCode, method)	Handles all child ID notifications at the child window.
EV_OWLDOCUMENT(id, method)	Handles new document notifications.
EV_OWLNOTIFY(id, method)	Generic document handler.
EV_OWLVIEW(id, method)	Handles view notifications.
EV_REGISTERED(str, method)	Handles registered MSG messages.

ID_xxxx file constants

inputdia.rh

These are the resource and control IDs for the input dialog box.

Table 1.15
ID file constants

Constant	Meaning
IDD_INPUTDIALOG	Resource ID number for the input dialog box.
ID_INPUT	Control ID for the user input.
ID_PROMPT	Control ID for the static text.

See also: *TInputDialog::SetUpWindow*

ID_XXXX printer constants**printer.rh**

The ID_XXXX printer constants are the resource and control IDs for the printer abort dialog box.

Table 1.16
ID printer constants

Constant	Meaning
IDD_ABORTDIALOG	Resource ID number for the abort dialog box.
ID_TITLE	Control ID for the selected printer driver.
ID_DEVICE	Control ID for the selected printer.
ID_PORT	Control ID for the selected printer port.

IDA_XXXX accelerator ID constants**editfile.rh**

IDA_EDITFILE is a resource ID for accelerator keys.

Table 1.17
Accelerator ID
constants

Constant	Meaning
IDA_EDITFILE	Resource ID for accelerator keys.

IDM_XXXX menu ID constant**editfile.rh**

IDM_EDITFILE is a resource ID for menu selections.

Table 1.18
Menu ID constants

Constant	Meaning
IDM_EDITFILE	Resource ID for menu selections.

IDS_XXXX document string ID constants**docview.rh**

Table 1.19
Document string ID
constants

Constant	Displays these messages:
IDS_DOCCHANGED	If the document has been changed, displays the message, "Do you want to save the changes?"
IDS_DOCLIST	Document is a document type.
IDS_NOTCHANGED	The document has not been changed.
IDS_UNABLECLOSE	Document manager is unable to close the document.
IDS_UNABLEOPEN	Document manager is unable to open the document.
IDS_UNTITLED	Document is untitled.
IDS_VIEWLIST	Document is a view type.

IDS_XXXX edit file ID constants**editfile.rh**

ObjectWindows defines these string constants used by edit and file classes to display information about files.

Table 1.20
Edit file ID constants

Constant	Meaning
IDS_FILECHANGED	The text in the file has changed. Do you want to save the changes?
IDS_FILEFILTER	Use this filter to display text files.
IDS_UNABLEREAD	Unable to read the file from the disk.
IDS_UNABLEWRITE	Unable to write the file to the disk.

IDS_XXXX exception messages**except.rh**

The following list includes general and application exception message constants grouped according to message types.

Table 1.21
Exception message constants

Constant	Meaning
IDS_INVALIDMAINWINDOW	Invalid MainWindow
IDS_INVALIDMODULE	Invalid module specified for window
IDS_NOAPP	No application object
IDS_OUTOFMEMORY	Out of memory
IDS_UNKNOWNERROR	Unknown error
IDS_MENUFAILURE	Menu creation failure
IDS_VALIDATORSYNTAX	Validator syntax error
IDS_PRINTERERROR	Printer error
Owl 1 compatibility messages:	
IDS_INVALIDCHILDWINDOW	Invalid child window
IDS_INVALIDCLIENTWINDOW	Invalid client window
IDS_INVALIDWINDOW	Invalid window
TXWindow messages:	
IDS_CHILDCREATEFAIL	Child create fail for window
IDS_CHILDREGISTERFAIL	Child class registration fails for window
IDS_CLASSREGISTERFAIL	Class registration fails for window
IDS_WINDOWCREATEFAIL	Create fail for window
IDS_WINDOWEXECUTEFAIL	Execute fail for window
GDI messages:	
IDS_GDIALLOCFAIL	GDI allocate failure
IDS_GDICREATEFAIL	GDI creation failure
IDS_GDIDELETEFAIL	GDI object delete failure
IDS_GDIDESTROYFAIL	GDI object destroy failure
IDS_GDIFAILURE	GDI failure
IDS_GDIFILEREADFAIL	GDI file read failure

Table 1.21: Exception message constants (continued)

IDS_INVALIDDIBHANDLE	Invalid DIB handle
IDS_GDIRESLOADFAIL	GDI resource load failure

IDS_XXXX listview ID constants**listview.rc**

ObjectWindows uses list view constants to define operations performed on views. These include clearing the document, inserting a new line, copying text to the Clipboard, and so on.

Table 1.22
Listview string ID
constants

Constant	Meaning
IDS_LISTVIEW	Resource ID for listview constants.

IDS_XXXX printer string ID constants**printer.rh**

ObjectWindows defines several constants used by printer classes to determine the printer status.

Table 1.23
Printer string ID
constants

Constant	String displayed
IDS_PRNCANCEL	Printing is canceled.
IDS_PRNERRORCAPTION	Printer error occurred.
IDS_PRNERRORTEMPLATE	Document was not printed.
IDS_PRNGENERORR	Error encountered during printing.
IDS_PRNMGRABORT	Printing aborted in Print Manager.
IDS_PRNON	Printer is on.
IDS_PRNOUTOFDISK	Out of disk space.
IDS_PRNOUTOFMEMORY	Out of memory.

IDS_XXXX validator ID constants**validator.rh**

ObjectWindows defines several constants used by validator classes to determine the validator status.

Table 1.24
Validator ID
constants

Constant	Meaning
IDS_VALPXPCONFORM	Item doesn't conform to correct picture format.
IDS_VALINVALIDCHAR	Character isn't one of the valid entries.
IDS_VALNOTINRANGE	Entry isn't within the specified range.
IDS_VALNOTINLIST	String isn't found in the list of valid entries.

IDW_MDICLIENT constant**framewin.h**

Child ID constant used to identify MDI client windows.

```
IDW_MDICLIENT
```

IDW_MDIFIRSTCHILD constant**framewin.h**

Child ID constant used to identify the first MDI client window.

```
IDW_FIRSTMDICHILD
```

lmParent constant**layoutco.h**

LmParent is used to construct layout metrics (for example, edge and size constraints).

```
#define lmParent 0
```

See also: *TLayoutConstraint*

LongMulDiv function**scroller.h**

TScroller uses this function to convert horizontal range values (*XRange*) from the scroll bar to horizontal scroll values (*XScrollValue*) and vice versa, or to convert vertical range values (*YRange*) from the scroll bar to vertical scroll values (*YScrollValue*) and vice versa.

```
inline long LongMulDiv(long mul1, long mul2, long div1);
```

See also: *TScroller*

MAX_RSRC_ERROR_STRING constant**except.h**

Maximum number of characters possible for an error message.

```
const int MAX_RSRC_ERROR_STRING = 255;
```

NBits function**color.h**

```
WORD NBits(int colors);
```

Returns the bit count corresponding to the given color count.

See also: *NColors*, *TColor* class

NColors function**color.h**

Returns the color count corresponding to the given bit count, or -1 if the bit count is not supported by Windows. Bit counts currently supported are 1, 4, 8, and 24.

```
int NColors(WORD bitCount);
```

See also: *NBits*, *TColor* class

ofxxxx document open enum**docview.h**

This enum defines the document and open sharing modes used for constructing streams and storing data. Any constants that have the same functionality as those used by OLE 2.0 docfiles are indicated in the following table; for example, *STGM_TRANSACTED*, *STGM_CONVERT*, *STGM_PRIORITY*, and *STGM_DELETEONRELEASE*.

Although files are typically used for data storage, databases or spreadsheets can also be used. I/O streams rather than DOS use these bit values. Documents open the object used for storage in one of the following modes:

Constant	Meaning
<i>ofParent</i>	A storage object is opened using the parent's mode.
<i>ofRead</i>	A storage object is opened for reading.
<i>ofWrite</i>	A storage object is opened for writing.
<i>ofReadWrite</i>	A storage object is opened for reading and writing.
<i>ofAtEnd</i>	Seek to end-of-file when opened originally.
<i>ofAppend</i>	Data is appended to the end of the storage object.
<i>ofTruncate</i>	An already existing file is truncated.
<i>ofNoCreate</i>	Open fails if file doesn't exist.
<i>ofNoReplace</i>	Open fails if file already exists.
<i>ofBinary</i>	Data is stored in a binary, not text, format. Carriage returns are not stripped.
<i>oflosMask</i>	All of the above bits are used by the ios class.

Constant	Meaning
ofTransacted	Changes to the storage object are preserved until the data is either committed to permanent storage or discarded. (STGM_TRANSACTED)
ofPreserve	Backs up previous storage data using before creating a new storage object with the same name. (STGM_CONVERT)
ofPriority	Supports temporary, efficient reading before opening the storage. (STGM_PRIORITY)
ofTemporary	The storage or stream is automatically destroyed when it is destructed. (STGM_DELETEONRELEASE)

See also: *TStream, InStream, OutStream*

pfxxxx property attribute constants

docview.h

These constants define document and view property attributes. Documents, views, and applications use these attributes to determine how to process a document or view.

Constant	Meaning
pfGetText	Property is accessible in a text format.
pfGetBinary	Property is accessible as a native nontext format.
pfConstant	Property can't be changed for the object instance.
pfSettable	Property can be set as a native format.
pfUnknown	Property is defined but unavailable for the object.
pfHidden	Property should be hidden from the user during normal browsing.
pfUserDef	Property has been user-defined at run time.

See also: *TDocument, TView*

BUILDOWLDLL macro

owldefs.h

`_BUILDOWLDLL`, which must be defined to build the ObjectWindows DLL, is used internally to control values for the `_OWLCLASS`, `_OWLDATA`, and `_OWLFUNC` macros. It is included in ObjectWindows makefiles to build the ObjectWindows DLL.

`_BUILDOWLDLL`

See also: `_OWLDLL`

_OWLCLASS macro

owldefs.h

Used internally by ObjectWindows, `_OWLCLASS` is the ObjectWindows version of `_RTLCLASS` adjusted to export and import WIN32 DLLs. Possible definitions for this macro are `__export`, `__far`, `__near`, `__tiny`, `__large`, or `__huge` model.

`_OWLCLASS`

_OWLDATA macro

owldefs.h

The macro `_OWLDATA` is the ObjectWindows version of `_RTLDATA` adjusted to export and import WIN32 DLLs for ObjectWindows. Possible definitions for this macro are `__export`, `__import`, or nothing.

`_OWLDATA`

_OWLDLL macro

owldefs.h

`_OWLDLL`, which is automatically defined if `_RTLDLL` is turned on, controls values for the `_OWLCLASS`, `_OWLDATA`, and `_OWLFUNC` macros. It must be defined if you are writing ObjectWindows applications or DLLs that use DLLs. This macro can also be turned on by a makefile.

`_OWLDLL`

_OWLFAR macro

owldefs.h

The macro `_OWLFAR` is the ObjectWindows version of `_RTLFAR` adapted to promote far data pointers in DLLs for ObjectWindows. Possible definitions for this macro are `__export` or `__import`.

`_OWLFAR`

_OWLFUNC macro

owldefs.h

The macro `_OWLFUNC` is ObjectWindows function version of `_RTLFUNC` adapted to export and import functions if building WIN32 DLLs for

ObjectWindows. Possible definitions for this macro are `__export`, `__import`, `__far`, or `__near`.

`_OWLFUNC`

OWLGetVersion function

owldefs.h

Returns the version number of the ObjectWindows library. The version number is represented as an unsigned short.

```
unsigned short far _OWLFUNC OWLGetVersion();
```

shxxxx document sharing enum

docview.h

The following file-sharing modes are available when opening document streams.

Table 1.25
shxxxx constants

Constant	Meaning
<code>shCompat</code>	Used for noncompliant applications, but should be avoided if possible.
<code>shNone</code>	<code>DENY_ALL</code> functionality.
<code>shRead</code>	<code>DENY_WRITE</code> functionality.
<code>shWrite</code>	<code>DENY_READ</code> functionality.
<code>shReadWrite</code>	<code>DENY_NONE</code> functionality.
<code>shDefault</code>	Use stream implementation default value.
<code>shMask</code>	<code>shCompat shNone shRead shWrite</code>

TActionFunc typedef

window.h

```
typedef void(*TActionFunc)(TWindow* win, void* param);
```

Passes a function pointer to `TWindow::ForEach`.

See also: `TWindow::ForEach`

TActionMemFunc typedef

window.h

```
typedef void(TWindow::*TActionMemFunc)(TWindow* win, void* param);
```

Passes a member function pointer to `TWindow::ForEach`.

See also: `TWindow::ForEach`

TAnyPMF typedef

dispatch.h

```
typedef void(GENERIC::*TAnyPMF)();
```

TAnyPMF is a generic pointer to a member function.

TAnyDispatcher typedef

dispatch.h

```
typedef LRESULT(*TAnyDispatcher)(GENERIC&, TAnyPMF, WPARAM, LPARAM);
```

TAnyDispatch is a message dispatcher type. All message dispatcher functions conform to this type and take four parameters:

- A reference to an object
- A pointer to the member function in which the signature varies according to the cracking that the function performs
- WPARAM
- LPARAM

TApplication class

applicat.h

Derived from *TModule*, *TApplication* acts as an object-oriented stand-in for a Windows application module. *TApplication* and *TModule* supply the basic behavior required of a Windows application. *TApplication* member functions create instances of a class, create main windows, and process messages. See Chapter 3 in the *Object Windows Programmer's Guide* for information about using application objects.

Public data members

cmdLine

```
const char far* cmdLine;
```

A null-terminated string, *cmdLine* points to a copy of the command-line arguments passed when the *TApplication* object is constructed. *cmdLine* is different from the WIN32 *cmdLine* in which the full path name of the module is appended to the command-line arguments. Whether running under WIN16 or WIN32, *ObjectWindows' TApplication::cmdLine* data member includes only the command-line arguments. Note that the runtime library global variables **_argv[]** and **_argc** contain identical information for both WIN16 and WIN32 APIs, and that **_argv[0]** points to the full path name of the module.

See also: *TApplication::TApplication*

cmdShow

```
int cmdShow;
```

nCmdShow indicates how the main window is to be displayed (either maximized or as an icon). These correspond to the *WinMain* parameter *CmdShow*. See *TApplication::nCmdShow* for a description of these constants.

See also: *TApplication::nCmdShow*

HAccTable

```
HACCEL HAccTable;
```

Included to provide backward compatibility, *HAccTable* holds a handle to the current Windows accelerator table being used by the application. New applications should instead use the accelerator table handle *TWindow*.

See also: *TWindow::LoadAcceleratorTable*, *TWindowAttr*

hPrevInstance

```
HINSTANCE hPrevInstance;
```

Contains the handle of the previously executing instance of the Windows application. If *hPrevInstance* is 0, there was no previously executing instance when this instance began execution. Under Win32, this value is always 0.

nCmdShow

```
int nCmdShow;
```

Indicates how the main window is to be displayed (either maximized or as an icon). These correspond to the *WinMain* parameter *CmdShow*. *nCmdShow* can contain one of the following Windows API constants:

Constant	Meaning
SW_SHOWDEFAULT	Shows the default SW_xxxx command.
SW_HIDE	Hides the window.
SW_MINIMIZE	Minimizes the specified window.
SW_SHOW	Activates a window using current size and position.
SW_SHOWMAXIMIZED	Displays a maximized window.
SW_SHOWMINIMIZED	Displays a minimized window.
SW_SHOWNA	Displays window in its current state.
SW_SHOWNOACTIVATE	Displays the window as an icon.
SW_SHOWNORMAL	Displays a window in its original size and position.
SW_SHOWSMOOTH	Shows a window by updating it in a bitmap and then copying the bits to the screen.

See also: *::ShowWindow*

Public constructors and destructor**Constructor**

```
TApplication(const char far* name = 0);
```

This *TApplication* constructor creates a *TApplication* object. Use this constructor in your *OwlMain* function.

Constructor

```
TApplication(const char far* name = 0, HINSTANCE instance,
             HINSTANCE prevInstance, const char far* cmdLine, int cmdShow,
             TModule* gModule = ::Module);
```

This *TApplication* constructor creates a *TApplication* object with the application name (*name*), the application instance handle (*instance*), the previous application instance handle (*prevInstance*), the command line invoked (*cmdLine*), and the number of main windows that should be shown (*cmdShow*). If you want to create your own *WinMain*, use this constructor because it provides access to the various arguments required by *WinMain*.

See also: *TApplication::nCmdShow*

Destructor

```
~TApplication();
```

~TApplication destroys the *TApplication* object.

Public member functions

BeginModal

```
int BeginModal(TWindow* window, int flags = MB_APPLMODAL);
```

BeginModal is called to begin a modal window's modal message loop. After determining which window to disable, *BeginModal* saves the current status of the window, disables the window, calls *MessageLoop*, and then reenables the window when the message loop is finished. The flags (which are the same as those passed to the Windows API function *MessageBox*) determine how *BeginModal* treats the window. *flags* can be one of the following values:

Constant	Meaning
MB_APPLMODAL	The window to be disabled (which is usually an ancestor of the modal window) is identified by <i>window</i> . If <i>window</i> is 0, no window is disabled.
MB_SYSTEMMODAL	The window to become system modal is identified by <i>window</i> .
MB_TASKMODAL	All top-level windows are disabled, and <i>window</i> is ignored.

BeginModal returns -1 if an error occurs.

See also: *::MessageBox*

BWCCEnabled

```
inline BOOL BWCCEnabled() const;
```

Indicates if the BWCC (Borland Custom Controls library) is enabled. Returns TRUE if BWCC is enabled or FALSE if BWCC is disabled.

CanClose

```
virtual BOOL CanClose();
```

Returns TRUE if it's OK for the application to close. By default, *CanClose* calls the *CanClose* member function of its main window and returns TRUE if both the main window and the document manager (*TDocManager*) can be closed. If any of the *CanClose* functions return FALSE, the application doesn't close. This member function is seldom redefined; closing behavior is usually redefined in the main window's *CanClose* member function, if needed.

See also: *TWindow::CanClose*, *TWindow::EvDestroy*

Condemn

```
void Condemn(TWindow* win);
```

Performs window cleanup.

Ctl3dEnabled

```
inline BOOL Ctl3dEnabled();
```

Returns TRUE if three-dimensional support (Microsoft 3-D Controls Library DLL) is enabled.

See also: *TApplication::EnableCtl3d*

EnableBWCC

```
void EnableBWCC(BOOL enable = TRUE, UINT Language = 0);
```

Loads and registers BWCC.DLL if you are running 16-bit applications or BWCC32.DLL if you are running 32-bit applications. By default, BWCC is enabled. To disable BWCC, set *enable* to FALSE.

See also: *TDialog*

EnableCtl3d

```
void EnableCtl3d(BOOL enable = TRUE);
```

Enables or disables the use of the CTL3D DLL. If *enable* is TRUE, *EnableCtl3d* loads and registers the CTL3D.DLL if it's not already enabled.

See also: *TApplication::Ctl3dEnabled*

EnableCtl3dAutosubclass void EnableCtl3dAutosubclass(BOOL enable);

Enables or disables CTL3D's use of autosubclassing if CTL3D is already enabled using *Ctl3dEnabled*. If autosubclassing is enabled, any non-ObjectWindows dialog boxes have a 3-D effect. The common dialog classes and *TDocManager* use this function to turn on autosubclassing before creating a non-ObjectWindows dialog box to make it three-dimensional and to turn off autosubclassing immediately after the dialog box is destroyed.

See also: *TDialog::EvCtlColor*

EndModal

```
void EndModal(int result);
```

EndModal is called to end a modal window's modal message loop. It uses the *BreakMessageLoop* flag to break the message loop and calls *MessageLoopResult* to pass the result.

Find

```
BOOL Find(TEventInfo &, TEqualOperator = 0);
```

Calls *TDocManager* to handle events. Delegates events as needed to the document manager.

See also: *TDocManager::Application*

GetDocManager

```
inline TDocManager* GetDocManager() const;
```

Returns a pointer to the document manager object that invoked the application.

See also: *TApplication::SetDocManager*

GetMainWindow

```
inline TFrameWindow* GetMainWindow(TFrameWindow* window);
```

Returns a pointer to the application's main window.

See also: *TApplication::SetMainWindow*

GetWinMainParams

```
inline void GetWinMainParams();
```

GetWinMainParams initializes a static instance of an application. ObjectWindows *OwlMain* uses this function to support static application instances.

See also: *TApplication::SetWinMainParams*

MessageLoop

```
virtual int MessageLoop();
```

Operates the application's message loop, which runs during the lifetime of the application. Queries Windows for messages; if one is received, *MessageLoop* processes it by calling *ProcessAppMsg*. If the query returns without a message, *MessageLoop* calls *IdleAction* to perform some processing during the idle time. *MessageLoop* calls *PumpWaitingMessages* to get and dispatch waiting messages. *MessageLoop* can be broken if *BreakMessageLoop* is set by *EndModal*.

See also: *TApplication::IdleAction*, *TApplication::ProcessAppMsg*

PostDispatchAction

```
void PostDispatchAction();
```

If *TApplication*'s message loop is not used, this function should be called after each message is dispatched

PreProcessMenu

```
virtual void PreProcessMenu(HMENU hmenu);
```

Your application can call *PreProcessMenu* to process the main window's menu before it is displayed.

See also: *TDocManager::EvPreProcessMenu*, *TMenu::TMenu*

ProcessAppMsg

```
virtual BOOL ProcessAppMsg(MSG& msg);
```

Checks for any special processing that is required for modeless dialog box, accelerator, and MDI accelerator messages. Calls the virtual *TWindow::PreProcessMsg* function of the window receiving the message. If your application does not create modeless dialog boxes, does not respond to accelerators, and is not an MDI application, you can improve performance by overriding this member function to return FALSE.

See also: *TWindow::PreProcessMsg*

PumpWaitingMessages `BOOL PumpWaitingMessages();`

Called by *MessageLoop*, *PumpWaitingMessages* processes and dispatches all waiting messages until the queue is empty. It also sets *BreakMessageLoop* when a WM_QUIT message is received.

QueryThrow

```
inline int QueryThrow();
```

QueryThrow tests to see if an exception is suspended and returns one or more of the bit flags in the *xs exception status enum*.

See also: *xs exception status enum*

ResumeThrow

```
void ResumeThrow();
```

ResumeThrow checks and rethrows suspended exceptions. Call this function any time you reenter ObjectWindows code from exception-unsafe code where an exception could have been thrown.

Run

```
virtual int Run();
```

Initializes the instance, calling *InitApplication* for the first executing instance and *InitInstance* for all instances. If the initialization is successful, *Run* calls *MessageLoop* and runs the application. If exceptions are thrown outside the message loop, *Run* catches these exceptions.

If an error occurs in the creation of a window, *Run* throws a *TXWindow* exception. If *Status* is assigned a nonzero value (which ObjectWindows uses to identify an error), a *TXCompatibility* exception is thrown.

See also: *TApplication::InitApplication*, *TApplication::InitInstance*, *TApplication::MessageLoop*

SetDocManager

```
TFrameWindow* SetDocManager(TDocManager* docManager);
```

Sets a pointer to the document manager object that invoked the application.

See also: *TApplication::GetDocManager*

SetMainWindow

```
TFrameWindow* SetMainWindow(TFrameWindow* window);
```

Sets up a new main window and sets the `WM_MAINWINDOW` flag so that the application knows this is the main window.

See also: *TApplication::GetMainWindow*

SetWinMainParams inline static void SetWinMainParams(HINSTANCE instance, HINSTANCE prevInstance, const char far* cmdLine, int cmdShow);

ObjectWindows default *WinMain* function calls *SetMainWinParams* so that *TApplication* can store the parameters for future use. To construct an application instance, *WinMain* calls the *OwlMain* function that's in the user's code. As it's being constructed, the application instance can fill in the parameters using those set earlier by *SetMainWinParams*.

See also: *TApplication::GetWinMainParams*

SuspendThrow void SuspendThrow(xalloc& x);

This version of *SuspendThrow* saves *xalloc* exception information.

SuspendThrow void SuspendThrow(xmsg& x);

This version of *SuspendThrow* saves *xmsg* exception information.

SuspendThrow void SuspendThrow(TXOwl& x);

This version of *SuspendThrow* saves a copy of a *TXOwl* exception.

SuspendThrow void SuspendThrow(int);

This version of *SuspendThrow* sets the *xs exception status* bit flags to the specified exception, for example *Bad_cast* or *Bad_typeid*.

See also: *xs exception status* enum

Protected data members

BreakMessageLoop BOOL BreakMessageLoop;

Causes the current modal message loop to break and terminate. If the current modal message loop is the main application, and your program sets *BreakMessageLoop*, your main application terminates.

See also: *TApplication::EndModal*, *TApplication::MessageLoop*, *TApplication::PumpWaitingMessages*

MessageLoopResult int MessageLoopResult;

MessageLoopResult is set by a call to *EndModal*. It contains the value that is returned by *MessageLoop* and *BeginModal*.

See also: *TApplication::BeginModal*, *TApplication::EndModal*,
TApplication::MessageLoop

Protected member functions

IdleAction

```
virtual void BOOL IdleAction(long idleCount);
```

ObjectWindows calls *IdleAction* when no messages are waiting to be processed. You can override *IdleAction* to do background processing. The default action is to give the main window a chance to do idle processing as long as *IdleAction* returns TRUE. *idleCount* specifies the number of times *IdleAction* has been called between messages.

See also: *TFrameWindow::IdleAction*

InitApplication

```
virtual void InitApplication();
```

ObjectWindows calls *InitApplication* to initialize the first instance of the application. For subsequent instances, this member function is not called.

The following sample program calls *InitApplication* the first time an instance of the program begins.

```
class TTestApp : public TApplication {
public:
    TTestApp(): TApplication("Instance Tester")
        {strcpy(WindowTitle, "Additional Instance");}

protected:
    char WindowTitle[20];

    void InitApplication() {strcpy(WindowTitle, "First Instance");}

    void InitMainWindow() {MainWindow = new TFrameWindow(0, WindowTitle);}
};

static TTestApp App;
```

InitInstance

```
virtual void InitInstance();
```

Performs each instance initialization necessary for the application. Unlike *InitApplication*, which is called for the first instance of an application, *InitInstance* is called whether or not there are other executing instances of the application. *InitInstance* calls *InitMainWindow*, and then creates and shows the main window element by *TWindow::Create* and *TWindow::Show*. If the main window can't be created, a *TXInvalidMainWindow* exception is thrown.



If you redefine this member function, be sure to explicitly call `TApplication::InitInstance`.

See also: `TApplication::InitApplication`, `TApplication::InitMainWindow`, `TApplication::Run`, `TModule::MakeWindow`, `TWindow::Show`

InitMainWindow

```
virtual void InitMainWindow();
```

By default, `InitMainWindow` constructs a generic `TFrameWindow` object with the name of the application as its caption. You can redefine `InitMainWindow` to construct a useful main window object of `TFrameWindow` (or a class derived from `TFrameWindow`) and store it in `MainWindow`. The main window must be a top-level window; that is, it must be derived from `TFrameWindow`. A typical use is

```
virtual void TMyApp::InitMainWindow(){
    MainWindow = new TMyWindow(NULL, Caption);
}
```

`InitMainWindow` can be overridden to construct a useful application.

TermInstance

```
virtual int TermInstance(int status);
```

Handles the termination of each executing instance of an ObjectWindows application.

TApplication::TXInvalidMainWindow class

applicat.h

A nested class, `TXInvalidMainWindow` describes an exception that results from an invalid Window. This exception is thrown if there is not enough memory to create a window or a dialog object. `InitInstance` throws this exception if it can't initialize an instance of an application object.

Public constructors

Constructor

```
TXInvalidMainWindow();
```

Constructs a `TXInvalidMainWindow` object with a default `IDS_INVALIDMAINWINDOW` message.

Public member functions

Clone

```
virtual TXOwl* Clone();
```


Makes a copy of the exception object. *Clone* must be implemented in any class derived from *TXOwl*.

Throw

```
virtual void Throw();
```

Throws the exception object. *Throw* must be implemented in any class derived from *TXOwl*.

TBandInfo class**dc.h**

An ObjectWindows **struct** equivalent to BANDINFOSTRUCT, *TBandInfo* is used to pass information to a printer driver that supports banding. *TBandInfo* is declared as follows:

```
struct TBandInfo {
    BOOL HasGraphics;
    BOOL HasText;
    TRect GraphicsRect;
};
```

HasGraphics is TRUE if graphics are (or are expected to be) on the page or in the band; otherwise, it is FALSE. *HasText* is TRUE if text is (or is expected to be) on the page or in the band; otherwise, it is FALSE. *GraphicsRect* defines the bounding region for all graphics on the page.

See also: *TPrintDC::BandInfo*, *TPrintDC::NextBand*

TBitmap class**gdiobjec.h**

TBitmap is the GDI bitmap class derived from *TGdiObject*. *TBitMap* can construct a bitmap from many sources. *TBitmap* objects are DDBs (device-dependent bitmaps), which are different from the DIBs (device-independent bitmaps) represented by *TDib* objects.

Public constructors**Constructor**

```
TBitmap(HBITMAP handle, TAutoDelete autoDelete = NoAutoDelete);
```

Creates a *TBitmap* object and sets the *Handle* data member to the given borrowed *handle*. The *ShouldDelete* data member defaults to FALSE, ensuring that the borrowed handle will not be deleted when the C++ object is destroyed.

See also: *TGdiObject::Handle*, *TGdiObject::ShouldDelete*

- Constructor** `TBitmap(const TClipboard& clipboard);`
Creates a *TBitmap* object with values from the given Clipboard.
See also: *TClipboard::GetClipboardData*
- Constructor** `TBitmap(const TBitmap& bitmap);`
Creates a copy of the given *bitmap* object.
See also: *TBitmap::GetObject*, *::CreateCompatibleBitmap*, *::CreateBitmap*
- Constructor** `TBitmap(int width, int height, BYTE planes, BYTE bitCount = 1,
void far* bits = 0);`
Creates a bitmap object from *bitCount* bits in the *bits* buffer with the given *width*, *height*, and *planes* argument values.
See also: *::CreateBitmap*
- Constructor** `TBitmap(const BITMAP far* bitmap);`
Creates a bitmap object with the values found in the given *bitmap* structure.
See also: *::CreateBitmapIndirect*, *struct BITMAP*
- Constructor** `TBitmap(const TDC& Dc, int width, int height, BOOL discardable = FALSE);`
Creates a bitmap object for the given device context with the given argument values.
See also: *::CreateDiscardableBitmap*, *TDC*
- Constructor** `TBitmap(const TDC& Dc, const TDib& dib, DWORD usage = CMB_INIT);`
Creates a bitmap object for the given device context with the given *dib* and *usage* argument values.
See also: *::CreateDIBBitmap*, *TDC*, *TDib*
- Constructor** `TBitmap(const TMetaFilePict& metaFile, TPalette& palette,
const TSize& size);`
Creates a bitmap object from the given *metaFile*, using the given *palette* and *size* arguments.
See also: *TPalette*
- Constructor** `TBitmap(const TDib& dib, const TPalette* palette = 0);`
Creates a bitmap object from the given *dib* and *palette* arguments. A working palette constructed from the DIB's color table is used if no palette is supplied. The default system palette can also be passed using *&TPalette::GetStock(TPalette::Default)*;

See also: *TScreenDC::RealizePalette*, *::CreateDIBBitmap*, *TDib*

Constructor

TBitmap(HINSTANCE instance, TResID resID);

Creates a bitmap object for the given application *instance* from the given resource.

See also: *::LoadBitmap*

Public member functions

BitsPixel

BYTE BitsPixel() const;

Returns the number of bits per pixel in this bitmap.

See also: *TBitmap::GetObject*

GetBitmapBits

inline DWORD GetBitmapBits(DWORD count, void far* bits) const;

Copies up to *count* bits of this bitmap to the buffer *bits*.

See also: *::GetBitmapBits*, *TBitmap::GetObject*,

GetBitmapDimension

inline BOOL GetBitmapDimension(TSize& size) const;

Retrieves the size of this bitmap (width and height, measured in tenths of millimeters) and sets it in the *size* argument. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *::GetBitmapDimensionEx*, *TBitmap::SetBitmapDimension*, *TSize*

GetObject

inline BOOL GetObject(BITMAP far& bitmap) const;

Retrieves data (width, height, and color format) for this bitmap and sets it in the given BITMAP structure. To retrieve the bit pattern, use *GetBitmapBits*.

See also: *TBitMap::GetBitmapBits*, struct *BITMAP*

Height

int Height() const;

Returns the height of this bitmap.

See also: *TBitmap::GetObject*

operator<<

inline TClipboard& operator<<(TClipboard& clipboard, TBitmap& bitmap);

Copies the given *bitmap* to the given *clipboard* argument. Returns a reference to the resulting Clipboard, which allows normal chaining of <<.

**operator
HBITMAP()**

inline operator HBITMAP() const;

Typecasting operator. Converts this bitmap's *Handle* to type *HBITMAP* (the Windows data type representing the handle to a physical bitmap).

Planes

```
BYTE Planes() const;
```

Returns the number of planes in this bitmap.

See also: *TBitmap::GetObject*

SetBitmapBits

```
inline DWORD SetBitmapBits(DWORD count, const void far* bits);
```

Copies up to *count* bits from the *bits* buffer to this bitmap.

See also: *::GetBitmapBits*

SetBitmapDimension

```
inline BOOL SetBitmapDimension(const TSize& size, TSize* oldSize=0);
```

Sets the size of this bitmap from the given *size* argument (width and height, measured in tenths of millimeters). The previous size is set in the *oldSize* argument. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *::SetBitmapDimensionEx*, *TBitmap::GetBitmapDimension*, *TSize*

ToClipboard

```
void ToClipboard(TClipboard& clipboard);
```

Copies this bitmap to the given Clipboard.

See also: *TClipboard::SetClipboardData*

Width

```
int Width() const;
```

Returns the width of this bitmap.

See also: *TBitmap::GetObject*

Protected member functions

HandleCreate

```
void HandleCreate(const TDib& dib, const TPalette &palette);
void HandleCreate(const TBitmap &src);
```

Creates a bitmap handle from the given argument objects.

TBitmapGadget class

bitmapga.h

Derived from *TGadget*, *TBitmapGadget* is a simple gadget that can display an array of bitmap images one at a time.

Public constructors and destructor

Constructor

```
TBitmapGadget(TResId bmpResId, int Id, TBorderStyle borderStyle,
              TResId bitmapName, int numImages, int startImage);
```

Constructs a *TBitmapGadget* and sets the current image to the beginning image in the array of images. Then, sets the border style to the current *TGadget* border style and *numImages* to the number of images in the array.

Destructor

```
~TBitmapGadget();
```

Deletes the array of images.

Public member functions

SelectImage

```
int SelectImage(int imageNum, BOOL immediate);
```

Determines the current image and repaints the client area if the image has changed. Calls the Windows API function `::UpdateWindow` to update the client area if the image has changed.

SysColorChange

```
void SysColorChange();
```

When the system colors have been changed, *SysColorChange* is called by the gadget window's *EvSysColorChange* so that bitmap gadgets can be rebuilt and repainted.

Protected member functions

GetDesiredSize

```
void GetDesiredSize(TSize& size);
```

Calls *TGadget::GetDesiredSize*, which determines how big the bitmap gadget can be. The gadget window sends this message to query the gadget's size. If shrink-wrapping is requested, *GetDesiredSize* returns the size needed to accommodate the borders and margins. If shrink-wrapping is not requested, it returns the gadget's current width and height. *TGadgetWindow* needs this information to determine how big the gadget needs to be, but it can adjust these dimensions if necessary. If *WideAsPossible* is TRUE, then the width parameter (*size.cx*) is ignored.

Paint

```
void Paint(TDC& dc);
```

Paints the gadget's border and the contents of the bitmap.

SetBounds

```
void SetBounds(TRect& r);
```

Calls *TGadget::SetBounds* and passes the dimensions of the bitmap gadget. *SetBounds* informs the control gadget of a change in its bounding rectangle.

See also: *TGadget::SetBounds*

TBitSet class

bitset.h

TBitSet sets or clears a single bit or a group of bits. You can use this class to set and clear option flags and to retrieve information about a set of bits. The class *TCharSet* performs similar operations for a string of characters.

Constructors

- | | |
|--------------------|--|
| Constructor | <code>TBitSet();</code>
Constructs a <i>TBitSet</i> object. |
| Constructor | <code>TBitSet(const TBitSet& bs);</code>
Constructs a <i>TBitSet</i> object as a copy of another <i>TBitSet</i> . |

Public member functions

- | | |
|--------------------|--|
| DisableItem | <code>void DisableItem(int item);</code>
Clears a single bit at <i>item</i> . |
| DisableItem | <code>void DisableItem(const TBitSet& bs);</code>
Clears the set of bits enabled in <i>bs</i> . |
| EnableItem | <code>void EnableItem(int item);</code>
Sets a single bit at <i>item</i> . |
| EnableItem | <code>void EnableItem(const TBitSet& bs);</code>
Sets the set of bits enabled in <i>bs</i> . |
| Has | <code>int Has(int item);</code>
Is nonzero if <i>item</i> is in the set of bits. |
| IsEmpty | <code>int TBitSet::IsEmpty();</code>
Is nonzero if the set is empty; otherwise, is 0. |
| operator += | <code>inline TBitSet& operator +=(int item);</code> |

Calls *EnableItem* to set a bit in the copied set. Returns a reference to the copied *TBitSet* object.

operator += inline TBitSet& operator +=(const TBitSet& bs);

Calls *EnableItem* to set the bits enabled in *bs*. Returns a reference to the copied *TBitSet* object.

operator -= inline TBitSet& operator -=(int item);

Calls *DisableItem* to clear a bit in the set. Returns a reference to the copied *TBitSet* object.

operator -= inline TBitSet& operator -=(const TBitSet& bs);

Calls *DisableItem* to clear the bits enabled in *bs*. Returns a reference to the copied *TBitSet* object.

operator &= TBitSet& operator &=(const TBitSet&);

ANDs all of the bits in the copied set and returns a reference to the copied *TBitSet* object.

operator |= TBitSet& operator |=(const TBitSet&);

ORs all of the bits in the copied set and returns a reference to the copied *TBitSet* object.

operator ~ TBitSet operator ~(const TBitSet&);

Returns the set of bits that is the opposite of a specified set of bits. For example, if the set of bits is 01010101, the returned set is 10101010. Returns a reference to the copied *TBitSet* object.

TBrush class

gdiobjec.h

The GDI Brush class is derived from *TGdiObject*. *TBrush* provides constructors for creating solid, styled, or patterned brushes from explicit information. It can also create a brush indirectly from a borrowed handle.

Public data members

enum StockId enum TStockId{Null, Black, DkGray, Gray, LtGray, White};

Enumerates the attributes of the stock brush objects.

Protected data members

Stocks[]

```
static TBrush Stocks[];
```

Note: This array no longer exists. Use `TDC::SelectStockObject` instead.

The single static array of Windows stock pen objects serving all *TBrush* objects. The stock brushes are `NULL_BRUSH`, `BLACK_BRUSH`, `DK_GRAY_BRUSH`, `GRAY_BRUSH`, `LTGRAY_BRUSH`, and `WHITE_BRUSH`.

Public constructors

Constructor

```
TBrush(HBRUSH handle, TAutoDelete autoDelete = NoAutoDelete);
```

Creates a *TBrush* object and sets the *Handle* data member to the given borrowed *handle*. The *ShouldDelete* data member defaults to `FALSE`, ensuring that the borrowed handle will not be deleted when the C++ object is destroyed.

See also: `TGdiObject::Handle`, `TGdiObject::ShouldDelete`

Constructor

```
TBrush(TColor color);
```

Creates a solid *TBrush* object with the given color. Sets *Handle* via a Win API `CreateSolidBrush(color)` call. To save a brush creation, this constructor uses a cache that can detect any color that matches a stock color.

See also: `::CreateSolidBrush`, `TColor`

Constructor

```
TBrush(TColor color, int style);
```

Creates a hatched *TBrush* object with the given style and color. Sets *Handle* via a Win API `CreateHatchedBrush(style, color)` call.

See also: `::CreateHatchedBrush`, `::CreateHashBrush`, `TColor`

Constructor

```
TBrush(const TBitmap& pattern);
```

Creates a patterned *TBrush* object with the given pattern. Sets *Handle* via a Win API `CreatePatternBrush(pattern)` call.

See also: `::CreatePatternBrush`

Constructor

```
TBrush(const TDib& pattern);
```

Creates a patterned *TBrush* object with the given DIB pattern. Sets *Handle* via a Win API `CreateDIBPatternBrush(pattern, pattern.usage())` call or the Win32 API `CreateDIBPatternBrushPt(pattern, pattern.usage())` call.

See also: `::CreateDIBPatternBrushPt`, `::CreateDIBPatternBrush`

Constructor

```
TBrush(const LOGBRUSH far* logBrush);
```

Creates a *TBrush* object with values from the given *logBrush*. Sets *Handle* via a Win API *CreateBrushIndirect(logBrush)* call.

See also: *::CreateBrushIndirect*

Public member functions

GetObject

```
inline BOOL GetObject(LOGBRUSH far& logBrush) const;
```

Retrieves information about this brush object and places it in the given LOGBRUSH structure. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *TGdiObject::GetObject*, struct *LOGBRUSH*

**operator
HBRUSH()**

```
inline operator HBRUSH() const;
```

Typecasting operator. Converts this brush's *Handle* to type *HBRUSH* (the Windows data type representing the handle to a physical brush).

UnrealizeObject

```
BOOL UnrealizeObject();
```

Directs the GDI to reset the origin of this brush the next time it is selected. Returns TRUE if call is successful; otherwise returns FALSE.

See also: *::UnrealizeObject*

TButton class**button.h**

TButton is an interface class that represents a push-button interface element in Windows. You must use a *TButton* to create a button control in a parent *TWindow*. You can also use a *TButton* to facilitate communication between your application and the button controls of a *TDialog*. This class is streamable.

There are two types of push buttons: regular and default. Regular buttons have a thin border. Default buttons (which represent the default action of the window) have a thick border.

Public data members

IsDefPB

BOOL IsDefPB;

Indicates whether the button is to be considered the default push button. Used for owner-draw buttons, *IsDefPB* is set by a *TButton* constructor based on BS_DEFPUSHBUTTON style setting.

Public constructors

Constructor

```
TButton(Window *parent, int Id, const char far *text, int X, int Y, int W,
        int H, BOOL isDefault = FALSE, TModule* module = 0);
```

Constructs a button object with the supplied parent window (*parent*), control ID (*Id*), associated text (*text*), position (*X*, *Y*) relative to the origin of the parent window's client area, width (*W*), and height (*H*). If *IsDefault* is TRUE, the constructor adds BS_DEFPUSHBUTTON to the default styles set for the *TButton* (in *Attr.Style*). Otherwise, it adds BS_PUSHBUTTON.

See also: *TControl::TControl*

Constructor

```
TButton(TWindow *parent, int resID, TModule* module = 0);
```

Constructs a *TButton* object to be associated with a button control of a *TDialog*. Calls *DisableTransfer* to exclude the button from the transfer mechanism because there is no data to be transferred.

The *resId* parameter must correspond to a button resource that you define.

See also: *TControl::TControl*

Protected data members

IsCurrentDefPB

BOOL IsCurrentDefPB;

Indicates whether the current button is the default push button.

Protected member functions

BMSetStyle

```
LRESULT BMSetStyle(WPARAM, LPARAM);
```

Because a button can't have both owner-drawn and push button styles, *BMSetStyle* keeps track of the style if the button is owner-drawn and Windows tries to set the style to BS_DEFPUSHBUTTON. *BMSetStyle* sets *IsCurrentDefPB* to TRUE if the button style is BS_DEFPUSHBUTTON.

- EvGetDlgCode** UINT EvGetDlgCode(MSG far*);
- Responds to WM_GETDLGCODE messages from the dialog manager. For owner-drawn buttons, *EvDlgCode* returns a code that indicates whether the button is the default button.
- See also: *TControl*
- GetClassName** char far* GetClassName();
- Returns the name of *TButton*'s Windows registration class, "BUTTON."
- SetupWindow** void SetupWindow();
- If the button is the default push button and an owner-drawn button, *SetupWindow* sends a DM_SETDEFID message to the parent window.

Response table entries

Response table entry	Member function
EV_WM_GETDLGCODE	EvGetDlgCode
EV_MESSAGE (BM_SETSTYLE, BMSetStyle)	BMSetStyle

TButtonGadget class

buttonga.h

Derived from *TGadget*, *TButtonGadgets* represent buttons that you can click on or off. You can also apply attributes such as color, style, and shape (notched or unnotched) to your button gadgets.

In general, button gadgets are classified as either command or attribute buttons. Attribute buttons include radio buttons (which are considered exclusive), or check boxes (which are nonexclusive). The public data member, *TType*, enumerates these button types.

TButtonGadget objects respond to mouse events in the following manner: when a mouse button is pressed, the button is pressed; when the mouse button is released, the button is released. Commands can be entered only when the mouse button is in the "up" state. When the mouse is pressed, *TButtonGadget* objects capture the mouse and reserve all mouse messages for the current window. When the mouse button is up, button gadgets release the capture for the current window. The public data member, *TState*, enumerates the three button states.

Public data members

- TShadowStyle**
enum
enum TShadowStyle;
Enumerates button shadow styles—either single (1) or double (2) shadow borders.
- TState** enum
enum TState;
TState enumerates the three button positions during which the button can be pressed: up (0), down (1), and an indeterminate state (2). A nonzero value indicates a highlighted button.
- TType** enum
enum TType;
Enumerates the types of buttons: command, exclusive, or nonexclusive.

Public constructors and destructor

- Constructor**
TButtonGadget(TResId uiBitmap, int id, TType type = Command,
 BOOL enabled = FALSE, TState state = Up,
 BOOL repeat = FALSE);
Constructs a *TButtonGadget* object using the specified bitmap ID, button gadget ID, and type, with the button enabled and in a button-up state.
See also: *TButtonGadget::State*
- Destructor**
~TButtonGadget();
Deconstructs a *TButtonGadget* object.

Public member functions

- CommandEnable**
void CommandEnable();
Enables the button gadget to capture messages. Calls *SendMessage* to send a WM_COMMAND_ENABLE message to the gadget window's parent, passing a *TCommandEnable: EvCommandEnable* message for this button.
- GetButtonState**
inline TState GetButtonState();
Returns the state of the button. If 0, the button is up, if 1, the button is down, if 2, the state is indeterminate.
See also: *TButtonGadget::State*

- GetButtonType** `inline TType GetButtonType();`
Returns the button type as 1 if the button is a command, 2 if exclusive, or 3 if nonexclusive.
- SetButtonState** `void SetButtonState(TState);`
Sets the state of the button. If the state has changed, the button is exclusive, and is in the down state, checks that the button is exclusive, sets *State*, and calls *Invalidate* to mark the changed area of the gadget for repainting.
See also: *TButtonGadget::State*
- SetNotchCorners** `void SetNotchCorners(BOOL);`
By default, *SetNotchCorners* implements notched corners for buttons. To repaint the frame of the button if the window has already been created, call *InvalidateRect* with the *Bounds* rectangle.
See also: *TButtonGadget::Invalidate*, *TGadget::InvalidateRect*, *TGadget::Paint*
- SetShadowStyle** `void SetShadowStyle(TShadowStyle);`
Sets the button style to a shadow style which, by default, is *DoubleShadow*. Sets the left and top borders to 2 and the right and bottom borders to *ShadowStyle + 1*.
- SysColorChange** `void SysColorChange();`
SysColorChange responds to an *EvSysColorChange* message forwarded by the owning *TGadgetWindow* by setting the dither brush to zero. If a user-interface bitmap exists, *SysColorchange* deletes and rebuilds it to get the new button colors.

Protected data members

- BitmapOrigin** `TPoint BitmapOrigin;`
Points to the x and y coordinates of the bitmap used for this button gadget.
- NotchCorners** `UINT NotchCorners;`
Initialized to 1, *NotchCorners* is 1 if the button gadget has notched corners or 0 if it doesn't have notched corners.
- Pressed** `UINT Pressed;`
Initialized to 1, *Pressed* is 1 if the button is released or 0 if it isn't released.

See also: *TButtonGadget::Activate*, *TButtonGadget::BeginPressed*, *TButtonGadget::CancelPressed*

Repeat	UINT Repeat; Initialized to 1, <i>Repeat</i> stores the repeat count for keyboard events.
ResId	TResId ResId; Holds the resource ID for this button gadget's bitmap.
ShadowStyle	TShadowStyle ShadowStyle; Holds the shadow style for the button—1 for single and 2 for double.
State	TState State; Holds the state of the button—either up, down, or indeterminate.
Type	TType Type; Holds the type of the button—either command, exclusive, or nonexclusive.
UIBitmap	TUIBitmap *UIBitmap; Holds the bitmap associated with the button gadget.

Protected member functions

Activate	virtual void Activate(TPoint& p); Invoked when the mouse is in the “up” state, <i>Activate</i> sets <i>Pressed</i> to FALSE, changes the state for attribute buttons, and paints the button in its current state. To do this, it calls <i>CancelPressed</i> , posts a WM_COMMAND message to the gadget window's parent, and sends menu messages to the gadget window's parent. See also: <i>TButtonGadget::Pressed</i>
BeginPressed	virtual void BeginPressed(TPoint& p); When the mouse button is pressed, <i>Beginpressed</i> sets <i>Pressed</i> to TRUE, paints the pressed button, and sends menu messages to the gadget window's parent. See also: <i>TButtonGadget::Pressed</i>
CancelPressed	virtual void CancelPressed(TPoint& p); When the mouse button is released, <i>CancelPressed</i> sets <i>Pressed</i> to FALSE, paints the button, and sends menu messages to the gadget window's parent.

See also: *TButtonGadget::Pressed*

GetDesiredSize void GetDesiredSize(TSize& size);

Stores the width and height (measured in pixels) of the button gadget in *size*. Calls *TGadget's GetDesiredSize* to calculate the relationship between one rectangle and another.

Invalidate void Invalidate();

If a button is pressed or the state of the button is changed, *Invalidate* invalidates (marks for repainting) the changed area of the gadget. *Invalidate* only invalidates the area that changes. To repaint the entire gadget, call *TGadget::InvalidateRect* and pass the rectangle's boundaries.

See also: *TButtonGadget::State*, *TGadget::InvalidateRect*

LButtonDown void LButtonDown(UINT modKeys, TPoint& p);

Overrides *TGadget* member function and responds to a left mouse button click by calling *BeginPressed*.

See also: *TButtonGadget::BeginPressed*

LButtonUp void LButtonUp(UINT modKeys, TPoint& p);

Overrides *TGadget* member functions and responds to a release of the left mouse button by calling *Activate*.

See also: *TButtonGadget::Activate*

MouseMove void Move(UINT modKeys, TPoint& p);

Calls *TGadget::MouseMove* in response to the mouse being dragged. If the mouse moves off the button, *MouseMove* calls *CancelPressed*. If the mouse moves back onto the button, *MouseMove* calls *BeginPressed*.

See also: *TButtonGadget::BeginPressed*, *TButtonGadget::CancelPressed*

Paint void Paint(TDC& dc);

Calls *::GetSystemMetrics* to get the width and height of the window frame (in pixels), calls *GetImageSize* to retrieve the size of the bitmap, and sets the inner rectangle to the specified dimensions. Calls *TGadget::PaintBorder* to perform the actual painting of the border of the control. Before painting the control, *Paint* determines whether the corners of the control are notched, and then calls *GetSysColor* to see if highlighting or shadow colors are used. *Paint* assumes the border style is plain. Then, *Paint* draws the top, left, right, and bottom of the control, adjusts the position of the bitmap, and finishes drawing the control using the specified embossing, fading, and dithering.

SetBounds `void SetBounds(TRect& r);`

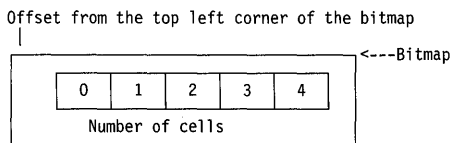
Gets the size of the bitmap, calls *TGadget::SetBounds* to set the boundary of the rectangle, and centers the bitmap within the button's rectangle.

See also: *TGadget::SetBounds*

TCelArray class

celarray.h

TCelArray is a horizontal array of cels (a unit of animation) created by slicing a portion of or an entire bitmap into evenly sized shapes. Gadgets such as buttons can use a *TCelArray* to save resource space. *TCelArray*'s functions let you control the dimensions of each cel and determine if the cel can delete the bitmap.



Public constructors and destructor

Constructor

```
TCelArray(TBitmap* bmp, int numCels, TSize celSize = 0, TPoint Offset = 0,
          TAutoDelete = AutoDelete);
```

Constructs a *TCelArray* from a bitmap by slicing the bitmap into a horizontal array of cels of a specified size. If *autoDelete* is TRUE, *TCelArray* can automatically delete the bitmap. The *ShouldDelete* data member defaults to TRUE, ensuring that the handle will be deleted when the bitmap is destroyed.

Constructor

```
TCelArray(TDib& dib, int numCels);
```

Constructs a *TCelArray* from a DIB (Device Independent Bitmap) by slicing the DIB into a horizontal array of evenly sized cels.

Constructor

```
TCelArray(const TCelArray& src);
```

Constructs a *TCelArray* as a copy of an existing one. If the original *TCelArray* owned its bitmap, the constructor copies this bitmap; otherwise, it keeps a reference to the bitmap.

Destructor

```
virtual ~TCelArray();
```


If *ShouldDelete* is TRUE (the default value), the bitmap is deleted. If *ShouldDelete* is FALSE, no action is taken.

Public member functions

CelSize	inline TSize CelSize() const; Returns the size in pixels of each cell.
CelOffset	inline TPoint CelOffset(TPoint offs); Returns the position of the upper left corner of a given cel relative to the upper left corner of the bitmap.
CelRect	TRect CelRect(int cel) const; Returns the upper left and lower right corner of a given cell relative to the upper left corner of the bitmap.
NumCels	inline int NumCels() const; Returns the number of cels in the array.
Offset	int Offset() const; Returns the offset of the entire <i>CelArray</i> .
operator []	inline TRect operator [] (int cel) const; Returns <i>CelRect</i> .
operator =	TCelArray& operator =(const TCelArray&); Returns <i>TCelArray</i> .
operator TBitmap&	inline operator TBitmap&(); Returns a reference to the bitmap.
SetCelSize	inline void SetCelSize(TSize size); Sets the size of each cel in the array.
SetOffset	inline void SetOffset(TPoint offs); Sets the offset for the cels in the array.
SetNumCels	inline void SetNumCels(int numCels); Sets the number of cels in the array.

Protected data members

Bitmap	TBitmap* Bitmap; Points to the bitmap.
CSize	TSize CSize; The size of a cell in the array.
Offs	TPoint Offs; Holds the offset of the upper left corner of the cel array from the upper left corner of the bitmap.
NCels	int NCels; The number of cells in the cel array.
ShouldDelete	BOOL ShouldDelete; Is TRUE if the destructor needs to delete the bitmap associated with the cel array.

TCharSet class

bitset.h

Derived from *TBitSet*, *TCharSet* sets and clears bytes for a group of characters. You can use this class to set or clear bits in a group of characters, such as the capital letters from "A" through "Z" or the lowercase letters from "a" through "z." The class *TBitSet* performs similar operations for a group of bits.

Public constructors

Constructor	TCharSet(); Constructs a <i>TCharSet</i> object.
Constructor	TCharSet(const TCharSet&); Copy constructor for a <i>TCharSet</i> object.
Constructor	TCharSet(const char far* str); Constructs a string of characters.

Public member functions

operator !=

```
inline int operator !=(const TBitSet& bs1, const TBitSet& bs2);
```

ORs all of the bits in the copied string and returns a reference to the copied *TCharSet* object.

TCheckBox class

checkbox.h

See Chapter 4 in the *ObjectWindows Programmer's Guide* for a description of interface objects. See Chapter 10 in the *ObjectWindows Programmer's Guide* for specific instructions about creating check box controls.

TCheckBox is a streamable interface class that represents a check box control. Use *TCheckBox* to create a check box control in a parent window. You can also use *TCheckBox* objects to more easily manipulate check boxes you created in a dialog box resource.

Two-state check boxes can be *checked* or *unchecked*; three-state check boxes have an additional *grayed* state. *TCheckBox* member functions let you easily control the check box's state. A check box can be in a group box (a *TGroupBox* object) that groups related controls.

Public data members

Group

```
TGroupBox *Group;
```

If the check box belongs to a group box (a *TGroupBox* object), *Group* points to that object. If the check box is not part of a group, *Group* is zero.

See also: *TGroupBox::TGroupBox*

Public constructors

Constructor

```
TCheckBox(TWindow *parent, int Id, const char far *title, int x, int y,
          int w, int h, TGroupBox *group, TModule* module = 0);
```

Constructs a check box object with the specified parent window (*parent*), control ID (*Id*), associated text (*title*), position relative to the origin of the parent window's client area (*x*, *y*), width (*w*), height (*h*), associated group box (*group*), and owning module (*module*). Invokes the *TButton* constructor with similar parameters. Sets the check box's *Attr.Style* to `WS_CHILD | WS_VISIBLE | WS_TABSTOP | BS_AUTOCHECKBOX`.

See also: *TButton::TButton*

Constructor

```
TCheckBox(TWindow *parent, int resourceId, TGroupBox *group, TModule*
          module = 0);
```

Constructs an associated *TCheckBox* object for the check box control with a resource ID of *resourceId* in the parent dialog box. Sets *Group* to *group* and then enables the data transfer mechanism by calling *EnableTransfer*.

See also: *TButton::TButton*, *TWindow::EnableTransfer*

Public member functions

Check

```
void Check();
```

Forces the check box to be checked by calling *SetCheck* with the value of *BF_CHECKED*. Notifies the associated group box, if there is one, that the state was changed.

See also: *TCheckBox::GetCheck*, *TCheckBox::Toggle*, *TCheckBox::Uncheck*, *TGroupBox::SelectionChanged*

GetCheck

```
inline UINT GetCheck() const;
```

Returns the state of the check box.

Table 1.26
TCheckBox check
states

Check box state	Return value
Checked	BF_CHECKED
Unchecked	BF_UNCHECKED
Grayed	BF_GRAYED

See also: *TCheckBox::SetCheck*

GetState

```
inline UINT GetState() const;
```

Returns the check, focus, and highlight state of the check box. See the *BM_GETSTATE* message in the Windows API online Help for more details.

See also: *TCheckBox::SetState*

SetCheck

```
void SetCheck(WORD check);
```

Forces the check box into the state specified by *Check*. See Table 1.26 for possible values of *Check*.

See also: *TCheckBox::GetCheck*

SetState

```
inline void SetState(UINT state);
```

Sets the check, focus, and highlight state of the check box. See the *BM_SETSTATE* message in the Windows API online Help for more details.

See also: *TCheckBox::GetState*

SetStyle

```
inline void SetStyle(UINT style, BOOL redraw);
```

Changes the style of the check box. See the `BM_SETSTYLE` message in the Windows API online Help for more details.

Toggle

```
void Toggle();
```

Toggles the check box between checked and unchecked if it's a two-state check box; toggles it between checked, unchecked, and gray if it's a three-state check box.

See also: *TCheckBox::SetCheck*

Transfer

```
virtual WORD Transfer(void *buffer, TTransferDirection direction);
```

Overrides *TWindow::Transfer*. Transfers the check state of the check box to or from *buffer*, using the values specified in Table 1.26. If *direction* is `tdGetData`, the check box state is transferred into the buffer. If *direction* is `tdSetData`, the check box state is changed to the settings in the transfer *buffer*.

Transfer returns the size of the transfer data in bytes. To get the size without actually transferring the check box, use *tdSizeData* as the *direction* argument.

Uncheck

```
inline void Uncheck();
```

Forces the check box to be unchecked by calling *SetCheck* with a value of `BF_UNCHECKED`. Notifies the associated group box, if there is one, that the state has changed.

See also: *TCheckBox::Check*, *TCheckBox::SetCheck*, *TCheckBox::Toggle*

Protected member functions

BNClicked

```
void BNClicked();
```

Responds to notification message `BN_CLICKED`, indicating that the user clicked the check box. If *Group* isn't 0, *BNClicked* calls the group box's *SelectionChanged* member function to notify the group box that the state has changed.

See also: *TGroupBox::SelectionChanged*

EvGetDlgCode

```
UINT EvGetDlgCode();
```

Overrides *TButton*'s response to the `WM_GETDLGCODE` message, an input procedure associated with a control that is not a check box, by calling *DefaultProcessing*.

See also: *TButton::EvGetDlgCode*, *TWindow::DefaultProcessing*

GetClassName

```
char far* GetClassName();
```

If BWCC is enabled, *TCheckBox* returns CHECK_CLASS. If BWCC is not enabled, returns "BUTTON."

Response table entries

Response table entry	Member function
EV_NOTIFY_AT_CHILD (BN_CLICKED, BNClicked)	BNClicked
EV_WM_GETDLGCODE	EVGetDlgCode

TChooseColorDialog class

chooseco.h

TChooseColorDialog objects represent modal dialog box interface elements that allow color selection and custom color adjustment. *TChooseColorDialog* can be made to appear modeless to the user by creating the dialog's parent as an invisible pop-up window and making the pop-up window a child of the main application window.

Public constructors

Constructor

```
TChooseColorDialog(TWindow* parent, TData& data, TResId templateID = 0,
                  const char far* title = 0, TModule* module = 0);
```

Constructs a dialog box with specified parent window, data, resource identifier, window caption, and library ID.

See also: *TChooseColorDialog::TData*

Public member functions

SetRGBColor

```
inline void SetRGBColor(TColor color);
```

Sets the current RGB color for the open dialog box by sending a *SetRGBMsgId*. You can use *SetRGBColor* to send a message to change the current color selection.

Protected data members

cc

```
CHOOSECOLOR cc;
```

Stores the length of the *TChooseColorDialog* structure, the window that owns the dialog box, and the data block that contains the dialog template. It also points to an array of 16 RGB values for the custom color boxes in the dialog box, and specifies the dialog-box initialization flags. This information is passed to the Windows API *ChooseColor* function.

See also: *TChooseColorDialog::TData*

Data

`TData& Data;`

Data is a reference to the *TData* object passed in the constructor.

See also: *TChooseColorDialog::TData*

SetRGBMsgId

`static UINT SetRGBMsgId;`

Contains the ID of the registered message sent by *SetRGBColor*.

Protected member functions

DialogFunction

`BOOL DialogFunction(UINT message, WPARAM, LPARAM);`

Returns TRUE if a message is handled.

See also: *TDialog::DialogFunction*

DoExecute

`int DoExecute();`

If no error occurs, *DoExecute* copies flags and colors into *Data* and returns zero. If an error occurs, *DoExecute* returns the IDCANCEL with *Data.Error* set to the value returned from *CommDlgExtendedError*.

EvSetRGBColor

`inline LPARAM EvSetRGBColor(WPARAM, LPARAM);`

Responds to the message sent by *SetRGBColor* by forwarding it to the original class via *DefaultProcessing*. This event handler is not in the response table.

Response table entries

The *TChooseColorDialog* response table has no entries.

TChooseColorDialog::TData struct

chooseco.h

The *TChooseColorDialog::TData* struct defines information necessary to initialize the dialog box with the user's color selection.

Public data members

Color

TColor Color;

Color specifies the color that is initially selected when the dialog box is created and contains the user's color selection when the dialog box is closed.

CustColors

TColor* CustColors;

CustColors points to an array of 16 colors.

Error

DWORD Error;

Error contains one of the following *CommDlgExtendedError* codes:

Constant	Meaning
CDERR_DIALOGFAILURE	Failed to create a dialog box.
CDERR_FINDRESFAILURE	Failed to find a specified resource.
CDERR_LOADRESFAILURE	Failed to load a specified resource.
CDERR_LOCKRESOURCEFAILURE	Failed to lock a specified resource.
CDERR_LOADSTRFAILURE	Failed to load a specified string.

Flags

DWORD Flags;

Flags can be a combination of the following Windows API constants:

Constant	Meaning
CC_FULLOPEN	Causes the entire dialog box to appear when the dialog box is created.
CC_PREVENTFULLOPEN	Disables the "Define Custom Colors" push button.
CC_RGBINIT	Causes the dialog box to use the color specified in <i>rgbResult</i> as the initial color selection.
CC_SHOWHELP	Causes the dialog box to show the Help push button.

See also: *TChooseColorDialog::Data*, *::ChooseColor*

TChooseFontDialog class

choosefo.h

A *TChooseFontDialog* represents modal dialog-box interface elements that create a system-defined dialog box from which the user can select a font, a font style (such as bold or italic), a point size, an effect (such as strikethrough or underline), and a color. *TChooseFontDialog* can be made to appear modeless by creating the dialog's parent as an invisible pop-up window and making the pop-up window a child of the main application window.

See Chapter 4 in the *ObjectWindows Programmer's Guide* for a general description of interface objects and Chapter 8 in the *ObjectWindows Programmer's Guide* for specific instructions about creating dialog boxes.

Public constructors

Constructor

```
TChooseFontDialog(TWindow* parent, TData& data, TResID templateID = 0,
                  const char far* title = 0, TModule* module = 0);
```

Constructs a dialog box with specified data, parent window, resource identifier, window caption, and library ID.

See also: *TChooseFontDialog::TData*

Protected data members

cf

```
CHOOSEFONT cf;
```

Contains font attributes that ObjectWindows passes to the Windows API *Choosefont* function. *cf* is initialized using fields in the *TChooseFontDialog::TData* class. It stores the length of the structure, the window that owns the dialog box and the data block that contains the dialog template. It also specifies the dialog-box initialization flags.

See also: *TChooseFontDialog::TData*

Data

```
TData& Data;
```

Data is a reference to the *TData* object passed in the constructor.

See also: *TChooseFontDialog::TData*

Protected member functions

CmFontApply

```
inline void CmFontApply();
```

Default handler for the third push button (the apply button) in the dialog box.

DialogFunction

```
BOOL DialogFunction(UINT message, WPARAM, LPARAM);
```

Returns TRUE if a message is handled.

See also: *TDialog::DialogFunction*

DoExecute

```
int DoExecute();
```

If no error occurs, *DoExecute* copies the flag values and font information into *Ddata*, and returns 0. If an error occurs, *DoExecute* returns the *CommDlgExtendedError*.

See also: *TChooseFontDialog::TData*

Response table entries

The *TChooseFontDialog* response table contains no entries.

TChooseFontDialog::TData struct

choosefo.h

The *ChooseFontDialog* structure defines information necessary to initialize the dialog box with the user's font selection.

Public data members

Color

TColor Color;

Indicates the font color that is initially selected when the dialog box is created; contains the user's font color selection when the dialog box is closed.

DC

HDC DC;

Handle to the device context from which fonts are obtained.

Error

DWORD Error;

Error contains one of the following *CommDlgExtendedError* codes:

Constant	Meaning
CDERR_DIALOGFAILURE	Failed to create a dialog box.
CDERR_FINDRESFAILURE	Failed to find a specified resource.
CDERR_LOCKRESOURCEFAILURE	Failed to lock a specified resource.
CDERR_LOADRESFAILURE	Failed to load a specified resource.
CDERR_LOADSTRFAILURE	Failed to load a specified string.
CFERR_MAXLESSTHANMIN	The size specified in <i>SizeMax</i> is less than the size in <i>SizeMin</i> .
CFERR_NOFONTS	No fonts exist.

Flags

DWORD Flags;

Flags can be a combination of the following Windows API constants:

Constant	Meaning
CF_APPLY	Enables the display and use of the Apply button.
CF_ANSIONLY	Specifies that the <i>ChooseFontDialog</i> structure allows only the selection of fonts that use the ANSI character set.
CF_BOTH	Causes the dialog box to list both the available printer and screen fonts.
CF_EFFECTS	Enables strikeout, underline, and color effects.
CF_FIXEDPITCHONLY	Enables fixed-pitch fonts only.
CF_FORCEFONTEXIST	Indicates an error if the user selects a nonexistent font or style.
CF_INITTOLOGFONTSTRUCT	Uses the LOGFONT structure at which <i>LogFont</i> points to initialize the dialog controls.
CF_LIMITSIZE	Limits font selection to those between <i>SizeMin</i> and <i>SizeMax</i> .
CF_NOSIMULATIONS	Does not allow GDI font simulations.
CF_PRINTERFONTS	Causes the dialog box to list only the fonts supported by the printer that is associated with the device context.
CF_SCALABLEONLY	Allows only the selection of scalable fonts.
CF_SCREENFONTS	Causes the dialog box to list only the system-supported screen fonts.
CF_SHOWHELP	Causes the dialog box to show the Help button.
CF_TTONLY	Enumerates and allows the selection of TrueType fonts only.
CF_USESTYLE	Specifies that <i>Style</i> points to a buffer containing the style attributes used to initialize the selection of font styles.
CF_WYSIWYG	Allows only the selection of fonts available on both the printer and the screen.

LogFont

LOGFONT LogFont;

Attributes of the font.

PointSize

int PointSize;

Point size of the font.

Style

char far* Style;

Style of the font such as bold, italic, underline, or strikeout.

FontType

WORD FontType;

Font type or name.

SizeMax

int SizeMax;

Maximum size of the font.

SizeMin

int SizeMin;

Minimum size of the font.

See also: *TChooseFontDialog::Data*

TClientDC class

dc.h

A DC class derived from *TWindowDC*, *TClientDC* provides access to the client area owned by a window.

Public constructors

Constructor

```
TClientDC(HWND wnd);
```

Creates a *TClientDC* object with the given owned window. The data member *Wnd* is set to *wnd*.

See also: *TWindowDC::Wnd*, *TDC::TDC*

TClipboard class

clipboard.h

TClipboard encapsulates several Windows API Clipboard functions that manipulate Clipboard data. You can open, close, empty, and paste data in a variety of data formats between the Clipboard and the open window.

Public data members

DefaultProtocol

```
static const char* DefaultProtocol;
```

Points to a string that specifies the name of the protocol the client needs. The default protocol is "StdFileEditing," which is the name of the object linking and embedding protocol. The macros `_OLE_H` or `_INC_OLE` must be defined before this function can be used.

See also: *TClipboard::QueryCreate*

Public member functions

CloseClipboard

```
inline void CloseClipboard();
```

Closes the Clipboard and returns TRUE if the Clipboard is closed or FALSE if it is not closed. Closing the Clipboard allows other applications to access the Clipboard.

See also: *TClipboard::CloseClipboard*, *::CloseClipboard*

CountClipboardFormats inline int CountClipboardFormats() const;

Returns a count of the number of types of data formats the Clipboard can use.

See also: *TClipboard::RegisterClipboardFormats*

EmptyClipboard inline BOOL EmptyClipboard();

Clears the Clipboard and frees any handles to the Clipboard's data. Returns TRUE if the Clipboard is empty, or FALSE if an error occurs.

See also: *::EmptyClipboard*

GetClipboardData inline HANDLE GetClipboardData(UINT format) const;

Retrieves data from the Clipboard in the format specified by *format*. For a description of Windows *CF_xxxx* data formats, see the Windows API online Help.

See also: *TClipboard::SetClipboardData*, *::GetClipboardData*

GetClipboardFormatName inline int GetClipboardFormatName(UINT format, char far* formatName, int maxCount) const;

Retrieves the name of the registered format specified by *format* and copies the format to the buffer pointed to by *formatName*. *maxCount* specifies the maximum length of the name of the format. If the name is longer than *maxCount*, it is truncated.

See also: *TClipboard::CountClipboardFormats*

GetClipboardOwner inline HWND GetClipboardOwner() const;

Retrieves the handle of the window that currently owns the Clipboard.

See also: *::GetClipboardOwner*

GetClipboardViewer inline HWND GetClipboardViewer() const;

Retrieves the handle of the first window in the Clipboard-view chain.

See also: *TClipboard::SetClipboardViewer*

GetOpenClipboardWindow HWND GetOpenClipboardWindow() const;

Retrieves the handle of the window that currently has the Clipboard open. If the Clipboard is not open, the return value is FALSE.

GetPriorityClipboardFormat inline int GetPriorityClipboardFormat(UINT FAR * priorityList, int count) const;

Returns the first Clipboard format in a list. *priorityList* points to an array that contains a list of the Clipboard formats arranged in order of priority. See the Windows API online Help for a description of these formats.

IsClipboardFormatAvailable inline BOOL IsClipboardFormatAvailable(UINT format) const;

Indicates if the format specified in *format* exists for use in the Clipboard. See the Windows API online Help for a description of Clipboard data formats.

See also: *::IsClipboardFormatAvailable*

OpenClipboard

inline BOOL OpenClipboard(HWND Wnd);

Opens the Clipboard and associates it with the window specified in *Wnd*.

See also: *TClipboard::CloseClipboard*, *::OpenClipboard*

operator BOOL

operator BOOL() const;

Checks handle. Should use *IsOk* instead.

QueryCreate

```
inline BOOL QueryCreate(const char far* protocol = DefaultProtocol,
                       OLEOPT_RENDER renderopt = olerender_draw,
                       OLECLIPFORMAT format = 0);
```

QueryCreate determines if the object on the Clipboard supports the specified protocol and rendering options. *DefaultProtocol* points to a string specifying the name of the protocol the client needs to use. *renderopt* specifies the client application's display and printing preference for the Clipboard object. *renderopt* is set to *olerender_draw*, which specifies that the client application calls the Windows API function *OleDraw*, which tells the client library to obtain and manage the data presentation. *format* specifies the Clipboard format the client application requests. The macros *_OLE_H* or *_INC_OLE* must be defined before this function can be used.

See the Windows API online Help for more information and a description of Clipboard formats.

See also: *TClipboard::QueryLink*

QueryLink

```
inline BOOL QueryLink(const char far* protocol = DefaultProtocol,
                     OLEOPT_RENDER renderopt = olerender_draw,
                     OLECLIPFORMAT format = 0);
```

QueryLink determines if a client application can use the Clipboard data to produce a linked object that uses the specified protocol and rendering options. See *TClipboard::QueryCreate* for a description of the parameters. The macros *_OLE_H* or *_INC_OLE* must be defined before this function can be used.

See the Windows API online Help for more information about this function.

See also: *TClipboard::QueryCreate*

RegisterClipboardFormat inline UINT RegisterClipboardFormat(const char far* formatName)
const;

Registers a new Clipboard format. *formatName* points to a character string that identifies the new format. If the format can't be registered, the return value is 0. See the Windows API online Help for more information.

See also: *TClipboard::CountClipboardFormats*

SetClipboardData inline HANDLE SetClipboardData(UINT format, HANDLE handle);

Sets a handle to the block of data at the location indicated by *handle*. *format* specifies the format of the data block. If successful, the return value is a handle to the data; if an error occurs, the return value is 0. See the Windows API online Help for more information.

See also: *TClipboard::GetClipboardData*, *::SetClipboardData*

SetClipboardViewer inline HWND SetClipboardViewer(HWND Wnd) const;

Adds the window specified by *Wnd* to the chain of windows that WM_DRAWCLIPBOARD notifies whenever the contents of the Clipboard change.

See also: *TClipboard::GetClipboardViewer*

Protected data members

IsOpen BOOL IsOpen;

Returns TRUE if the Clipboard is open.

TheClipboard static TClipboard TheClipboard;

Protected constructors and destructor

Constructor TClipboard();

Constructs a *TClipboard* object.

Destructor ~TClipboard();

Destroys a *TClipboard* object.

TClipboardViewer class

clipview.h

Registers a *TClipboardViewer* as a Clipboard viewer when the user interface element is created, and removes itself from the Clipboard-viewer chain when it is destroyed.

Public constructors

- Constructor** `TClipboardViewer();`
Constructs a *TClipboardView* object.
- Constructor** `TClipboardViewer(HWND hWnd, TModule* module = 0);`
Constructs a *TClipboardViewer* object with a handle (*hWnd*) to the windows that will receive notification when the Clipboard's contents are changed.

Public member functions

- EvChangeCBChain** `void EvChangeCBChain(HWND hWndRemoved, HWND hWndNext);`
Responds to a Windows API `WM_CHANGECHAIN` message. *hWndRemoved* is a handle to the window that's being removed. *hWndNext* is the window following the removed window.
- EvDestroy** `void EvDestroy();`
Responds to a Windows API `WM_DESTROY` message when a window is removed from the Clipboard-viewer chain.
- EvDrawClipboard** `void EvDrawClipboard();`
Responds to a Windows API `WM_DRAWCLIPBOARD` message sent to the window in the Clipboard-viewer chain when the contents of the Clipboard change.
- SetupWindow** `void SetupWindow();`
Adds a window to the Clipboard-viewer chain.
See also: *TWindow::SetupWindow*

Protected data members

HWNDNext

HWND HWNDNext;

Specifies the next window in the Clipboard-viewer chain.

Response table entries

Response table entry	Member function
EV_WM_CHANGECHAIN	EvChangeCbChain
EV_WM_DESTROY	EvDestroy
EV_WM_DRAWCLIPBOARD	EvDrawClipBoard

TColor class

color.h

TColor is a support class used in conjunction with the classes *TPalette*, *TPaletteEntry*, *TRgbQuad*, and *TRgbTriple* to simplify all Windows color operations. *TColor* has ten static data members representing the standard RGB COLORREF values, from *Black* to *White*. Constructors are provided to create *TColor* objects from COLORREF and RGB values, palette indexes, palette entries, and RGBQUAD and RGBTRIPLE values. See the entries for *NBits* and *NColors* for a description of *TColor*-related functions.

Public data members

Black

static const TColor Black;

The static *TColor* object with fixed *Value* set by RGB(0, 0, 0). See the RGB Windows macro for more information.

Gray

static const TColor Gray;

Contains the static *TColor* object with fixed *Value* set by RGB(128, 128, 128).

LtBlue

static const TColor LtBlue;

Contains the static *TColor* object with the fixed *Value* set by RGB(0, 0, 255).

LtCyan

static const TColor LtCyan;

Contains the static *TColor* object with the fixed *Value* set by RGB(0, 255, 255).

LtGray

static const TColor LtGray;

Contains the static *TColor* object with the fixed *Value* set by RGB(192, 192, 192).

LtGreen `static const TColor LtGreen;`

Contains the static *TColor* object with the fixed *Value* set by RGB(0, 255, 0).

LtMagenta `static const TColor LtMagenta;`

Contains the static *TColor* object with the fixed *Value* set by RGB(255, 0, 255).

LtRed `static const TColor LtRed;`

Contains the static *TColor* object with the fixed *Value* set by RGB(255, 0, 0).

LtYellow `static const TColor LtYellow;`

Contains the static *TColor* object with the fixed *Value* set by RGB(255, 255, 0).

White `static const TColor White;`

Contains the static *TColor* object with the fixed *Value* set by RGB(255, 255, 255).

See also: Windows macro RGB

Public constructors

Constructor `TColor();`

The default constructor sets *Value* to 0.

See also: *TColor::Value*

Constructor `TColor(COLORREF value);`

Creates a *TColor* object with *Value* set to the given *value*.

See also: *TColor::Value*

Constructor `TColor(long value);`

`TColor(long value) : Value((COLORREF)value) {}`

Creates a *TColor* object with *Value* set to *(COLORREF)value*.

See also: *TColor::Value*, COLORREF

Constructor `TColor(int r, int g, int b);`

Creates a *TColor* object with *Value* set to RGB(*r,g,b*).

See also: *TColor::Value*, RGB macro

Constructor

`TColor(int r, int g, int b, int f);`

Creates a *TColor* object with *Value* set to *RGB(r,g,b)* with the flag byte formed from *f*.

See also: *TColor::Value*, PALETTEENTRY struct

Constructor

`TColor(int index);`

Creates a *TColor* object with *Value* set to *PALETTEINDEX(index)*.

See also: *TColor::Value*, PALETTEINDEX macro

Constructor

`TColor(const PALETTEENTRY far& pe);`

Creates a *TColor* object with *Value* set to:

`RGB(pe.peRed, pe.peGreen, pe.peBlue)`

See also: *TColor::Value*, RGB macro, PALETTEENTRY struct

Constructor

`TColor(const RGBQUAD far& q);`

Creates a *TColor* object with *Value* set to:

`RGB(q.rgbRed, q.rgbGreen, q.rgbBlue)`

See also: *TColor::Value*, RGB macro, RGBQUAD struct

Constructor

`TColor(const RGBTRIPLE far& t);`

Creates a *TColor* object with *Value* set to:

`RGB(t.rgbtRed, t.rgbtGreen, t.rgbtBlue)`

See also: *TColor::Value*, RGB macro, RGBTRIPLE struct

Public member functions

Blue

`inline BYTE Blue() const;`

Returns the blue component of this color's *Value*.

See also: *TColor::Red*, *TColor::Green*, RGB

Flags

`inline BYTE Flags() const;`

Returns the *peFlags* value of this object's *Value*.

See also: *TPaletteEntry*

Green

`inline BYTE Green() const;`

Returns the green component of this color's *Value*.

See also: *TColor::Red*, *TColor::Blue*, RGB

operator==

```
inline BOOL operator==(const TColor& clrVal);
```

Returns TRUE if this color's *Value* equals *clrValue*; otherwise returns FALSE.

See also: *TColor::Value*

**operator
COLORREF()**

```
inline operator COLORREF() const;
```

Type-conversion operator that returns *Value*.

See also: *TColor::Value*

Index

```
inline int Index() const;
```

Returns the index value corresponding to this color's *Value* by masking out the two upper bytes. Used when color is a palette index value.

See also: *TColor::Value*, COLORREF

PalIndex

```
inline TColor PalIndex() const;
```

Returns the palette index corresponding to this color's *Value*. The returned color has the high-order byte set to 1.

See also: *TColor::Value*, *TColor::Index*, COLORREF

PalRelative

```
inline TColor PalRelative() const;
```

Returns the palette-relative RGB corresponding to this color's *Value*. The returned color has the high-order byte set to 2.

See also: *TColor::Value*, *TColor::Rgb*, COLORREF

Red

```
inline BYTE Red() const;
```

Returns the red component of this color's *Value*.

See also: *TColor::Blue*, *TColor::Green*, RGB

Rgb

```
inline TColor Rgb() const;
```

Returns the explicit RGB color corresponding to this color's *Value* by masking out the high-order byte.

See also: *TColor::Value*, COLORREF

Protected data members**Value**

```
COLORREF Value;
```

The color value of this *TColor* object. *Value* can have three different forms, depending on the application:

- Explicit values for RGB (red, green, blue)
- An index into a logical color palette
- A palette-relative RGB value

TComboBox class

combobox.h

You can use *TComboBox* to create a combo box or a combo box control in a parent *TWindow*, or to facilitate communication between your application and the combo-box controls of *TDialog*. *TComboBox* objects inherit most of their behavior from *TListBox*. This class is streamable.

There are three types of combo boxes: simple, drop down, and drop down list. These types are governed by the style constants *CBS_SIMPLE*, *CBS_DROPDOWN*, and *CBS_DROPDOWNLIST*. These constants, supplied to the constructor of a *TComboBox*, tell Windows the type of combo box element to create.

Public data members

TextLen

WORD TextLen;

Contains the length of the text in the combo box's associated edit control.

Public constructors

Constructor

```
TComboBox(TWindow *parent, int Id, int x, int y, int w, int h,
          DWORD style, WORD textLen, TModule* module = 0);
```

Constructs a combo box object with the specified parent window (*parent*), control ID (*Id*), position (*x, y*) relative to the origin of the parent window's client area, width (*w*), height (*h*), style (*style*), and text length (*textLen*).

Invokes the *TListBox* constructor with similar parameters. Then sets *Attr.Style* as follows:

```
Attr.Style = WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP | CBS_SORT |
            CBS_AUTOHSCROLL | WS_VSCROLL | style;
```

One of the following combo box style constants must be among the styles set in *style*: *CBS_SIMPLE*, *CBS_DROPDOWN*, *CBS_DROPDOWNLIST*, *CBS_OWNERDRAWFIXED*, or *CBS_OWNERDRAWVARIABLE*.

See also: *TComboBox::TextLen*, *TListBox::TListBox*

Constructor

```
TComboBox(TWindow* parent, int Id, WORD textLen, TModule* module = 0);
```

Constructs a default combo box with the given parent window, control ID, and text length.

Public member functions

AddString

```
inline virtual int AddString(const char far* string);
```

Adds a string to an associated list part of a combo box. Returns the index of the string in the list. The first entry is at index zero. Returns a negative value if an error occurs.

Clear

```
inline void Clear();
```

Clears the text of the associated edit control.

ClearList

```
inline virtual void ClearList();
```

Clears out all associated entries in the associated list.

DeleteString

```
inline virtual int DeleteString(int index);
```

Deletes the string at the passed index in the associated list part of a combo box. Returns a count of the entries remaining in the list or a negative value if an error occurs.

DirectoryList

```
inline virtual int DirectoryList(UINT attrs, const char far* fileSpec);
```

Fills the combo box with file names from a specified directory.

FindString

```
inline virtual int FindString(const char far* find, int indexStart) const;
```

Searches for a match beginning at the passed Index. If a match is not found after the last string has been compared, the search continues from the beginning of the list until a match is found or until the list has been completely traversed. Returns the index of the first string in the associated list part of a combo box or a negative value if an error occurs.

GetCount

```
inline virtual int GetCount() const;
```

Returns the number of entries in the associated list part of the combo box or a negative value if an error occurs.

```
GetDroppedControlRect inline void GetDroppedControlRect(TRect& Rect) const;
```

For combo boxes, gets the screen coordinates of the dropped down list box.

GetDroppedState

```
inline BOOL GetDroppedState() const;
```

For drop down combo boxes, determines if a list box is visible.

GetEditSel

```
int GetEditSel(int &startPos, int &endPos);
```

Returns the starting and ending positions of the text selected in the associated edit control. Returns `CB_ERR` if the combo box has no edit control.

GetExtendedUI

```
inline BOOL GetExtendedUI() const;
```

Determines if the combo box has the extended user interface, which differs from the default user interface in the following ways:

- Displays the list box if the user clicks the static text field.
- Displays the list box if the user presses the ↓ key.
- Disables scrolling in the static text field if the item list is not visible.

Returns `TRUE` if the combo box has the extended user interface; otherwise returns `FALSE`.

See also: *TComboBox::SetExtendedUI*

GetItemData

```
inline virtual DWORD GetItemData(int index) const;
```

Returns the 32-bit value associated with the combo box's item.

See also: *TListBox::GetItemData*

GetItemHeight

```
inline int GetItemHeight(int index) const;
```

Returns the height in pixels of the Combo box's list items or `CB_ERR` if an error occurs.

See also: *TComboBox::GetItemData*, *TListBox::GetItemData*

GetSelIndex

```
inline virtual int GetSelIndex() const;
```

Returns the index of the list selection or a negative value if none exists.

GetString

```
inline virtual int GetString(char far* str, int index) const;
```

Retrieves the contents of the string at the position supplied in *index* and returns it in *string*. *GetString* returns the string length or a negative value if an error occurs. The buffer must be large enough for the string and the terminating zero.

See also: *TListBox::GetString*

GetStringLen

```
inline virtual int GetStringLen(int index) const;
```

Returns the string length (excluding the terminating zero) of the item at the position *index* supplied in *index*. Returns a negative value if an error occurs.

See also: *TListBox::GetStringLen*

GetText

```
inline int GetText(char far* str, int maxChars) const;
```

Retrieves the number of characters in the edit or static portion of the combo box.

GetTextLen

```
inline int GetTextLen() const;
```

Returns the text length (excluding the terminating zero) of the edit control or static portion of the combo box.

HideList

```
inline void HideList();
```

Hides the drop down list of a drop down or drop down list combo box.

InsertString

```
inline virtual int InsertString(const char far* str, int index);
```

Inserts a string in the associated list part of a combo box at the position supplied in *Index*. Returns the index of the string in the list or a negative value if an error occurs.

See also: *TListBox::InsertString*

SetEditSel

```
inline int SetEditSel(int startPos, int endPos);
```

Selects characters that are between *startPos* and *endPos* in the edit control of the combo box. Returns `CB_ERR` if the combo box does not have an edit control.

SetExtendedUI

```
inline int SetExtendedUI(BOOL extended);
```

If the combo box has the extended user interface, sets the extended user interface.

See also: *TComboBox::GetExtendedUI*

SetItemData

```
inline virtual int SetItemData(int index, DWORD data);
```

Sets the 32-bit value associated with the *TComboBox*'s item. Returns `CB_ERR` if an error occurs.

SetItemHeight

```
inline int SetItemHeight(int index, int height);
```

Sets the height of the list items or the edit control portion in a combo box. If the index or height is invalid, returns `CB_ERR`.

See also: *TComboBox::GetItemHeight*

SetSelIndex

```
inline virtual int SetSelIndex(int index);
```

Sets the index of the list selection.

See also: *TComboBox::GetSelIndex*

SetSelString inline virtual int SetSelString(const char far* findStr, int indexStart);
Selects a string of characters in the associated list box and sets the contents of the associated edit control to the supplied string.

SetText void SetText(const char far* string);
Selects the first string in the associated list box that begins with the supplied *string*. If there is no match, *SetText* sets the contents of the associated edit control to the supplied string and selects it.

ShowList inline void ShowList();
Shows the list of a drop down or drop down list combo box.
See also: *TComboBox::HideList*

ShowList void ShowList(BOOL show);
Returns TRUE if the list is displayed.

Transfer WORD Transfer(void* buffer, TTransferDirection direction);
Transfers the items and selection of the combo box to or from a transfer buffer if *tdSetData* or *tdGetData*, respectively, is passed as the *direction*. *buffer* is expected to point to a *TComboBoxData* structure.
Transfer returns the size of a pointer to a *TComboBoxData*. To retrieve the size without transferring data, your application must pass *tdSizeData* as the *direction*.



You must use a pointer in your transfer buffer to these structures. You cannot embed copies of the structures in your transfer buffer, and you cannot use these structures as transfer buffers.

See also: *TComboBoxData*, *TWindow::Transfer*

Protected member functions

GetClassName virtual char far* GetClassName();
Returns the name of *TComboBox*'s Windows registration class, "ComboBox."

SetupWindow void SetupWindow();
Sets up the window and limits the amount of text the user can enter in the combo box's edit control to the value of *TextLen* minus 1.

See also: *TWindow::SetupWindow*

TComboBoxData class**combobox.h**

A *TComboBoxData* is an interface object that represents a transfer buffer for a *TComboBox*.

Public data members

ItemDatas	<pre>TDwordArray* ItemDatas;</pre> <p>Array of DWORDs to transfer into and out of the combo box's associated list box.</p> <p>See also: <i>TComboBoxData::SetItemData</i>, <i>TComboBoxData::GetItemData</i></p>
Selection	<pre>char* Selection;</pre> <p>Points to the currently selected string to transfer to or from a combo box.</p>
SelIndex	<pre>int SelIndex;</pre> <p>Index of the selected item in the strings array. If zero, the index is used to transfer to. If negative, no item is selected.</p>
Strings	<pre>TStringArray* Strings;</pre> <p>Array of class string to transfer in or out of the combo box's associated list box.</p>

Public constructors and destructor

Constructor	<pre>TComboBoxData();</pre> <p>Constructs a <i>TComboBox</i> object, initializes <i>Strings</i> and <i>ItemDatas</i> to empty arrays, and initializes <i>Selection</i> and <i>SelIndex</i> to 0.</p>
Destructor	<pre>~TComboBoxData();</pre> <p>Deletes <i>Strings</i>, <i>ItemDatas</i>, and <i>Selection</i>.</p>

Public member functions

AddItemData	<pre>inline void AddItemData(DWORD itemData);</pre> <p>Adds the user-defined item data to the <i>ItemDatas</i> array.</p>
AddString	<pre>void AddString(const char *str, BOOL isSelected = FALSE);</pre>

Adds the specified string to the array of *Strings*. If *IsSelected* is TRUE, *AddString* deletes *Selection* and copies *string* into *Selection*.

AddStringItem

```
inline void AddStringItem(const char* str, DWORD itemData,
                          BOOL isSelected = FALSE);
```

Calls *AddItemData* to add the item data to the *ItemDatas* array, and calls *AddString* to add a string to the array of *Strings*.

TCommonDialog class**commddial.h**

Derived from *TDialog*, *TCommonDialog* is the abstract base class for *TCommonDialog* objects. It provides the basic functionality for creating dialog boxes using the common dialog DLL.

Public constructors**Constructor**

```
TCommonDialog(TWindow* parent, const char far* title = 0, TModule*
              module = 0);
```

Invokes a *TWindow* constructor, passing *parent* and *library ID*. Constructs a common dialog box.

Public member functions**DoCreate**

```
HWND DoCreate();
```

Called by *Create*, *DoCreate* creates a modeless dialog box. It returns 0 if unsuccessful.

See also: *TDialog::Create*

DoExecute

```
int DoExecute();
```

Called by *Execute*, *DoExecute* creates a modal dialog box. It returns IDCANCEL if canceled or unsuccessful.

See also: *TDialog::Execute*

Protected data members**CDTitle**

```
const char far* CDTitle;
```

CDTitle stores the optional caption displayed in the common dialog box.

See also: *TDialog::SetCaption*

Protected member functions

CmHelp

```
inline void CmHelp();
```

Default handler for the *pshHelp* push button (the Help button in the dialog box).

CmOkCancel

```
inline void CmOkCancel();
```

Responds to a click on the dialog box's OK or Cancel button by calling *DefaultProcessing* to let the common dialog DLL process the command.

See also: *TDialog::Cancel*, *TDialog::Ok*

EvClose

```
inline void EvClose();
```

Responds to a WM_CLOSE message by calling *DefaultProcessing* to let the common dialog DLL process the command.

See also: *TDialog::EvClose*

SetupWindow

```
void SetupWindow();
```

Assigns the caption of the dialog box to *CDTitle* if *CDTitle* is nonzero.

See also: *TDialog::SetupWindow*

Response table entries

Response table entry	Member function
EV_COMMAND(IDCANCEL, CmOkCancel)	CmOkCancel
EV_COMMAND(IDOK, CmOkCancel)	CmOkCancel
EV_WM_CLOSE	EvClose
EV_WM_CTLCOLOR	EvCtlColor

TCondFunc type

window.h

Defines a member function type used by *TWindow*'s function *FirstThat*.

```
typedef BOOL (*TCondFunc) (TWindow *win, void *param);
```

See also: *TWindow::FirstThat*

TCondMemFunc typedef**window.h**

Defines a member function type used by *TWindow's* function *FirstThat*.

```
typedef BOOL (TWindow::TCondMemFunc) (*win, void *param);
```

See also: *TWindow::FirstThat*

TControl class**control.h**

TControl unifies its derived control classes, such as *TScrollBar*, *TControlGadget*, and *TButton*. Control objects of derived classes are used to represent control interface elements in Windows. A control object must be used to create a control in a parent *TWindow* or a derived window. A control object can be used to facilitate communication between your application and the controls of a *TDialog*. *TControl* is a streamable class.

Public constructors**Constructor**

```
TControl(TWindow *parent, int Id, const char far *title, int x, int y,
         int w, int h, TModule* module = 0);
```

Invokes *TWindow's* constructor, passing it *parent* (parent window), *title* (caption text), and *module*. Sets *Attr* using the supplied library ID (*Id*), position (*x, y*) relative to the origin of the parent window's client area, width (*w*), and height (*h*) parameters. It sets *Attr.Style* to `WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP`.

See also: *TWindow::TWindow*

Constructor

```
TControl(TWindow *parent, int resourceId, TModule* module = 0);
```

Constructs an object to be associated with an interface control of a *TDialog*. Invokes the *TWindow* constructor, and then enables the data transfer mechanism by calling *EnableTransfer*.

The *ResId* parameter must correspond to a control interface resource that you define.

See also: *TWindow::TWindow*, *TWindow::EnableTransfer*

Public member functions**CompareItem**

```
virtual int CompareItem (COMPAREITEMSTRUCT far &);
```

Used in owner-draw combo boxes, *CompareItem* compares two items. The derived class supplies the compare logic.

DeleteItem

```
virtual void DeleteItem (DELETEITEMSTRUCT far &);
```

DeleteItem is used in owner-draw combo boxes. In such cases, the derived class supplies the delete logic.

DrawItem

```
virtual void DrawItem(DRAWITEMSTRUCT far &drawInfo);
```

DrawItem responds to a message forwarded to a drawable control by *TWindow* when the control needs to be drawn. *TControl::DrawItem* calls *ODADrawEntire* if the entire control needs to be drawn, calls *ODASelect* if the selection state of the control has changed, or calls *ODAFocus* if the focus has been shifted to or from the control.

See also: *TControl::ODADrawEntire*, *TControl::ODASelect*, *TControl::ODAFocus*, *TWindow::EvDrawItem*

MeasureItem

```
virtual void MeasureItem (MEASUREITEMSTRUCT far &);
```

Used by owner-drawn controls to inform Windows of the dimensions of the specified item. For list boxes and control boxes, this function applies to specific items; for other owner-drawn controls, this function is used to inform Windows of the total size of the control.

Protected member functions

EvPaint

```
void EvPaint();
```

If the control has a predefined Windows class, *EvPaint* calls *TWindow::DefaultProcesing* for Windows-supplied painting. Otherwise, it calls *TWindow::EvPaint*.

See also: *TWindow::DefaultProcesing*, *TWindow::EvPaint*

ODADrawEntire

```
virtual void ODADrawEntire(DRAWITEMSTRUCT far &drawInfo);
```

Responds to a notification message sent to a drawable control when the control needs to be drawn. *ODADrawEntire* can be redefined by a drawable control to specify the manner in which it is to be drawn.

See also: *TWindow::EvDrawItem*, *TControl::DrawItem*

ODAFocus

```
virtual void ODAFocus(DRAWITEMSTRUCT far &drawInfo);
```

Responds to a notification sent to a drawable control when the focus has shifted to or from the control. *ODAFocus* can be redefined by a drawable control to specify the manner in which it is to be drawn when losing or gaining the focus.

See also: *TWindow::EvDrawItem*, *TControl::DrawItem*

ODASelect

```
virtual void ODASelect(DRAWITEMSTRUCT far &drawInfo);
```

Responds to a notification sent to a drawable control when the selection state of the control changes. By default, *ODASelect* calls *Parent->DrawItem*. *ODASelect* can be redefined by a drawable control to specify the manner in which it is drawn when its selection state changes.

See also: *TWindow::EvDrawItem*, *DrawItem*

Response table entries

Response table entry	Member function
EV_WM_PAINT	EvPaint

TControlBar class

controlb.h

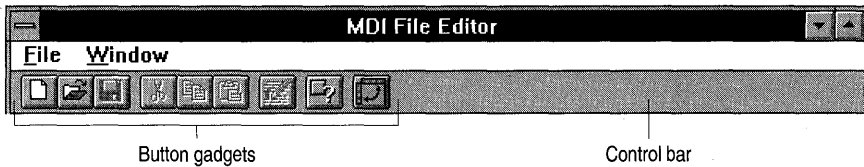
Derived from *TGadgetWindow*, *TControlBar* implements a control bar that provides mnemonic access for its button gadgets. To construct, build, and insert a control bar into a frame window, you can first define the following response table:

```
DEFINE_RESPONSE_TABLE1(TMDIFileApp, TApplication)
    EV_COMMAND(CM_FILENEW, CmFileNew),
    EV_COMMAND(CM_FILEOPEN, CmFileOpen),
    EV_COMMAND(CM_SAVESTATE, CmSaveState),
    EV_COMMAND(CM_RESTORESTATE, CmRestoreState),
END_RESPONSE_TABLE;
```

Next, add statements that will construct a main window and load its menu, accelerator table, and icon. Then, to construct, build and insert a control bar into the frame window, insert these statements:

```
TControlBar* cb = new TControlBar(frame);
cb->Insert(*new TButtonGadget(CM_FILENEW, CM_FILENEW));
cb->Insert(*new TButtonGadget(CM_FILEOPEN, CM_FILEOPEN));
cb->Insert(*new TButtonGadget(CM_FILESAVE, CM_FILESAVE));
cb->Insert(*new TSeparatorGadget(6));
cb->Insert(*new TButtonGadget(CM_EDITCUT, CM_EDITCUT));
cb->Insert(*new TButtonGadget(CM_EDITCOPY, CM_EDITCOPY));
cb->Insert(*new TButtonGadget(CM_EDITPASTE, CM_EDITPASTE));
cb->Insert(*new TSeparatorGadget(6));
cb->Insert(*new TButtonGadget(CM_EDITUNDO, CM_EDITUNDO));
frame->Insert(*cb, TDecoratedFrame::Top);
```

The sample MDIFILE.CPP ObjectWindows program on your distribution disk displays the following control bar:



Public constructors

Constructor

```
TControlBar(TWindow* parent = 0, TTileDirection direction = Horizontal,
            TFont* font = new TGadgetWindowFont, TModule* module = 0);
```

Constructs a *TControlBar* interface object with the specified direction (either horizontal or vertical) and window font.

Public member functions

PreProcessMsg

```
BOOL PreProcessMsg(MSG& msg);
```

Performs preprocessing of Windows messages. Because *PreProcessMsg* does not translate any accelerator keys for *TControlBar*, it returns FALSE.

Protected member functions

PositionGadget

```
void PositionGadget(TGadget* previous, TGadget* next, TPoint& p);
```

Gets the border style, determines the direction of the gadget, and positions the button gadget on either a horizontal or vertical border if any overlapping is required.

TControlGadget class

controlg.h

TControlGadget serves as a surrogate for *TControl* so that you can place *TControl* objects such as edit controls, buttons, sliders, gauges, or VBX controls, into a gadget window. If necessary, *TControlGadget* sets a parent window and creates the control gadget. See *TGadget* for more information about gadget objects.

Public constructors and destructor

- Constructor** `TControlGadget(TWindow& control, TBorderStyle = None);`
 Creates a *TControlGadget* object associated with the specified *TControl* window.
- Destructor** `~TControlGadget();`
 Destroys a *TControlGadget* object and removes it from the associated window.

Protected data members

- Control** `TWindow* Control;`
 Points to the control window that is managed by this *TControlGadget*.

Protected member functions

- GetDesiredSize** `void GetDesiredSize(TSize& size);`
 Calls *TGadget::GetDesiredSize* and passes the *size* of the control gadget.
 See also: *TGadget::GetDesiredSize*
- GetInnerRect** `void GetInnerRect(TRect&);`
 Computes the area of the control gadget's rectangle excluding the borders and margins.
- Inserted** `void Inserted();`
 Called when the control gadget is inserted in the parent window. Calls *::ShowWindow* to display the window in its current size and position.
- Invalidate** `void Invalidate(BOOL erase = TRUE);`
 Used to invalidate the active (usually nonborder) portion of the control gadget, *Invalidate* calls *InvalidateRect* and passes the boundary width and height of the area to erase.
- InvalidateRect** `void InvalidateRect(const TRect&, BOOL erase = TRUE);`
 Invalidates the control gadget rectangle in the parent window.
- Removed** `void Removed();`
 Called when the control gadget is removed from the parent window.

SetBounds

```
void SetBounds(TRect& rect);
```

Calls *TGadget::SetBounds* and passes the dimensions of the control gadget's rectangle. *SetBounds* informs the control gadget of a change in its bounding rectangle.

See also: *TGadget::SetBounds*

Update

```
void Update();
```

Calls the Windows API function *::UpdateWindow* to update the client area of the specified window by sending a WM_PAINT message immediately.

Response table entries

The *TControlGadget* class has no response table entries.

TCreatedDC class

dc.h

An abstract *TDC* class, *TCreatedDC* serves as the base for DCs that are created and deleted. *TCreatedDC* does much of the work of creating and deleting HDCs using *CreateDC* and *::DeleteDC*. See *TDC* for more information about DC objects.

Public constructors and destructor

Constructor

```
TCreatedDC(const char far* driver, const char far* device,
           const char far* output, const DEVMODE far* initData=0);
```

Creates a DC object for the device specified by *driver* (driver name), *device* (device name), and *output* (the name of the DOS file or device [port] for the physical output medium). The optional *initData* argument provides a *DEVMODE* structure containing device-specific initialization data for this DC. *initData* must be 0 (the default) if the device is to use any default initializations specified by the user via the Control Panel.

See also: *::CreateDC*, *::ExtDeviceMode*

Constructor

```
TCreatedDC(HDC handle, TAutoDelete autoDelete);
```

Creates a DC object using an existing DC.

See also: *enum TDC::TAutoDelete*, *TDC::ShouldDelete*

Destructor

```
~TCreatedDC();
```

Calls *RestoreObjects* and clears any nonzero *OrgXXX* data members. If *ShouldDelete* is TRUE, the destructor deletes this DC via *::DeleteDC*.

See also: *TDC::ShouldDelete*, *::DeleteDC*, *TDC::RestoreObjects*

Protected member functions

TCreatedDC

`TCreatedDC();`

Creates a device context for the given device. DC objects can be constructed either by borrowing an existing HDC handle or by supplying device and driver information.

See also: *::CreateDC*

TCursor class

gdiobjec.h

TCursor, derived from *TGdiobject*, represents the GDI cursor object class. *TCursor* constructors can create cursors from a resource or from explicit information. Because cursors are not real GDI objects, the *TCursor* destructor overloads the base destructor, *~TGdiObject()*.

Public constructors and destructor

Constructor

`TCursor(HCURSOR handle, TAutoDelete autoDelete = NoAutoDelete);`

Creates a *TCursor* object and sets the *Handle* data member to the given borrowed *handle*. The *ShouldDelete* data member defaults to FALSE, ensuring that the borrowed handle will not be deleted when the C++ object is destroyed.

See also: *TGdiObject::Handle*, *TGdiObject::ShouldDelete*

Constructor

`TCursor(HINSTANCE instance, const TCursor& cursor);`

Creates a copy of the given *cursor* object by calling the Win API function *CopyCursor(instance, cursor)*.

See also: *::CopyIcon*, *::CopyCursor*

Constructor

`TCursor(HINSTANCE instance, TResID resID);`

Constructs a *cursor* object from the specified resource ID.

See also: *::LoadCursor*

Constructor

```
TCursor(HINSTANCE instance, const TPoint& hotSpot, const TSize& size,
        void far* andBits, void far* xorBits);
```



Constructs a *TCursor* object of the specified size and at the specified point.

See also: *::CreateCursor*, *TPoint*, *TSize*

Constructor

```
TCursor(const void* resBits, DWORD resSize);
```



Constructs a *TCursor* object from the specified resource.

See also: *::CreateIconFromResource*

Constructor

```
TCursor(const ICONINFO* iconInfo);
```

Creates a *TCursor* object from the specified *ICONINFO* structure information.

See also: *::CreateIconIndirect*

Destructor

```
virtual ~TCursor();
```

Destroys a *TCursor* object.

See also: *~TGdiObject*

Public member functions

**operator
HCURSOR()**

```
operator HCURSOR() const;
```

An inline typecasting operator. Converts this cursor's *Handle* to type *HCURSOR* (the Windows data type representing the handle to a cursor resource).

GetIconInfo

```
inline BOOL GetIconInfo(ICONINFO* iconInfo) const;
```



Retrieves information about this icon and copies it in the given *ICONINFO* structure. Returns *TRUE* if the call is successful; otherwise returns *FALSE*.

See also: *::GetIconInfo*, *struct ICONINFO*

TDC class

dc.h

TDC is the root class for GDI DC wrappers. Each *TDC* object has a *Handle* protected data member of type *HDC* (handle to DC). Win API functions that take an *HDC* argument can therefore be called by a corresponding *TDC* member function without this explicit handle argument.

DC objects can be created directly with *TDC* constructors, or via the constructors of specialized subclasses (such as *TWindowDC*, *TMemoryDC*, *TMetaFileDC*, *TDibDC*, and *TPrintDC*) to get specific behavior. DC objects can be constructed with an already existing and borrowed HDC handle or from scratch by supplying device/driver information as with `::CreateDC()`. The class *TCreateDC* takes over much of the creation and deletion work from *TDC*.

TDC has four handles as protected data members: *OrgBrush*, *OrgPen*, *OrgFont*, and *OrgPalette*. These handles keep track of the stock GDI objects selected into each DC. As new GDI objects are selected with *SelectObject()* or *SelectPalette()*, these data members store the previous objects. The latter can be restored individually with *RestoreBrush*, *RestorePen()*, and so on, or they can all be restored with *RestoreObjects()*. When a *TDC* object is destroyed (via `~TDC::TDC`), all the originally selected objects are restored. The data member *TDC::ShouldDelete* controls the deletion of the underlying Windows DC.

Public data members

enum TAutoDelete

```
enum TAutoDelete{NoAutoDelete, AutoDelete};
```

Flag for handle constructors to control GDI object deletion in the *TDC* destructor.

See also: *TDC::ShouldDelete*, `~TDC`

Public constructors and destructor

Constructor

```
TDC(HDC handle, TAutoDelete autoDelete = NoAutoDelete);
```

Creates a DC object “borrowing” the handle of an existing DC. The *Handle* data member is set to the given *handle* argument.

See also: *enum TDC::TAutoDelete*, *TDC::ShouldDelete*

Constructor

```
TDC(const char far* driver, const char far* device,  
    const char far* output, const DEVMODE far* initData = 0);
```

Creates a DC object for the device specified by *driver* (driver name), *device* (device name), and *output* (the name of the DOS file or device [port] for the physical output medium). The optional *initData* argument provides a *DEVMODE* structure containing device-specific initialization data for this DC. *initData* must be 0 (the default) if the device is to use any default initializations specified by the user via the Control Panel.

See also: `::CreateDC`, `::ExtDeviceMode`, `TCreatedDC`

Destructor

```
virtual ~TDC();
```

Calls `RestoreObjects` and clears any nonzero `OrgXXX` data members. If `ShouldDelete` is TRUE, the destructor deletes this DC via `::DeleteDC`.

See also: `TDC::ShouldDelete`, `::DeleteDC`, `TDC::RestoreObjects`

Public member functions

AngleArc



```
inline BOOL AngleArc(int x, int y, DWORD radius, float startAngle,
                    float sweepAngle);
```

```
inline BOOL AngleArc(const TPoint& center, DWORD radius, float startAngle,
                    float sweepAngle);
```

Draws a line segment and an arc on this DC using the currently selected pen object. The line is drawn from the current position to the beginning of the arc. The arc is that part of the circle (with the center at logical coordinates (x, y) and positive radius, *radius*) starting at *startAngle* and ending at $(startAngle + sweepAngle)$. Both angles are measured in degrees, counterclockwise (the default arc direction) from the x-axis. The arc might appear to be elliptical, depending on the current transformation and mapping mode. `AngleArc` returns TRUE if the figure is drawn successfully; otherwise, it returns FALSE. If successful, the current position is moved to the end point of the arc.

See also: `::AngleArc`, `TDC::Arc`, `::SetArcDirection`

Arc

```
inline BOOL Arc(int x1, int y1, int x2, int y2, int x3, int y3, int x4,
               int y4);
```

```
inline BOOL Arc(const TRect& r, const TPoint& start, const TPoint& end);
```

Draws an elliptical arc on this DC using the currently selected pen object. The center of the arc is the center of the bounding rectangle, specified either by $(x1, y1)/(x2, y2)$ or by the rectangle *r*. The starting/ending points of the arc are specified either by $(x3, y3)/(x4, y4)$ or by the points *start* and *end*. All points are specified in logical coordinates. `Arc` returns TRUE if the arc is drawn successfully; otherwise, it returns FALSE. The current position is neither used nor altered by this call. The drawing direction is set by `::SetArcDirection`; the default is counterclockwise.

See also: `::Arc`, `::SetArcDirection`, `TDC::AngleArc`

BeginPath



```
inline BOOL BeginPath();
```

Opens a new path bracket for this DC and discards any previous paths from this DC. Once a path bracket is open, an application can start calling

draw functions on this DC to define the points that lie within that path. The draw functions that define points in a path are the following TDC members: *AngleArc*, *Arc*, *Chord*, *CloseFigure*, *Ellipse*, *ExtTextOut*, *LineTo*, *MoveToEx*, *Pie*, *PolyBezier*, *PolyBezierTo*, *PolyDraw*, *Polygon*, *Polyline*, *PolylineTo*, *PolyPolygon*, *PolyPolyline*, *Rectangle*, *RoundRect*, and *TextOut*.

A path bracket can be closed by calling *TDC::EndPath*.

BeginPath returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: *::BeginPath*, *TDC::FillPath*, *TDC::EndPath*, *TDC::PathToRegion*, *TDC::StrokePath*, *TDC::StrokeandFillPath*, *TDC::WidenPath*

BitBlt

```
inline BOOL BitBlt(int dstX, int dstY, int w, int h, const TDC& srcDC,
                  int srcX, int srcY, DWORD rop);
inline BOOL BitBlt(const TRect& dst, const TDC& srcDC, const TPoint& src,
                  DWORD rop);
```

Performs a bit-block transfer from *srcDc* (the given source DC) to this DC (the destination DC). Color bits are copied from a source rectangle to a destination rectangle. The location of the source rectangle is specified either by its upper left-corner logical coordinates (*srcX*, *srcY*), or by the *TPoint* object, *src*. The destination rectangle can be specified either by its upper left-corner logical coordinates (*dstX*, *dstY*), width *w*, and height *h*, or by the *TRect* object, *dst*. The destination rectangle has the same width and height as the source. The *rop* argument specifies the raster operation used to combine the color data for each pixel. See *TDC::MaskBlt* for a detailed list of *rop* codes.

When recording an enhanced metafile, an error occurs if the source DC identifies the enhanced metafile DC.

If necessary, *PlgBlt* adjusts the source color formats to match those of the destination. Before using *PlgBlt*, an application should call *GetDeviceCaps* to determine if the source and destination DCs are compatible. If they are incompatible, an error occurs.

See also: *::BitBlt*

Chord

```
inline BOOL Chord(int x1, int y1, int x2, int y2, int x3, int y3, int x4,
                  int y4);
inline BOOL Chord(const TRect& R, const TPoint& Start, const TPoint& End);
```

Draws and fills a chord (a region bounded by the intersection of an ellipse and a line segment) on this DC using the currently selected pen and brush objects. The ellipse is specified by a bounding rectangle given either by (*x1*, *y1*)/(*x2*, *y2*) or by the rectangle *R*. The starting/ending points of the chord

are specified either by $(x3, y3)/(x4, y4)$ or by the points *Start* and *End*. *Chord* returns TRUE if the call is successful; otherwise, it returns FALSE. The current position is neither used nor altered by this call.

See also: `::Chord`, `TDC::Arc`

CloseFigure



```
inline BOOL CloseFigure();
```

Closes an open figure in this DC's open path bracket by drawing a line from the current position to the first point of the figure (usually the point specified by the most recent `TDC::MoveTo` call), and connecting the lines using the current join style for this DC. If you close a figure with `TDC::LineTo` instead of with `CloseFigure`, end caps (instead of a join) are used to create the corner. The call fails if there is no open path bracket on this DC. Any line or curve added to the path after a `CloseFigure` call starts a new figure. A figure in a path remains open until it is explicitly closed with `CloseFigure` even if its current position and start point happen to coincide.

`CloseFigure` returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: `::CloseFigure`, `TDC::BeginPath`, `TDC::EndPath`

DPtoLP

```
inline BOOL DPtoLP(TPoint* points, int count = 1) const;
```

Converts each of the *count* points in the *points* array from device points to logical points. The conversion depends on this DC's current mapping mode and the settings of its window and viewport origins and extents. `DPtoLP` returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: `TDC::LPtoDP`, `::DPtoLP`

DrawFocusRect

```
inline BOOL DrawFocusRect(int x1, int x2, int y1, int y2);
inline BOOL DrawFocusRect(const TRect& rect);
```

Draws the given rectangle on this DC in the style used to indicate focus. Calling the function a second time with the same *rect* argument will remove the rectangle from the display. A rectangle drawn with `DrawFocusRect` cannot be scrolled. `DrawFocusRect` returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: `::DrawFocusRect`

DrawIcon

```
inline BOOL DrawIcon(int x, int y, const TIcon& icon);
inline BOOL DrawIcon(const TPoint& point, const TIcon& icon);
```

Draws the given *icon* on this DC. The upper left corner of the drawn icon can be specified by *x*- and *y*-coordinates or by the *point* argument. `DrawIcon` returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: `::DrawIcon`

DrawText

```
inline virtual BOOL DrawText(const char far* string, int count,
                             const TRect& r, WORD format = 0);
```

Formats and draws in the given rectangle, *r*, up to *count* characters of the null-terminated *string* using the current font for this DC. If *count* is -1, the whole string is written. The rectangle must be specified in logical units. Formatting is controlled with the *format* argument, which can be various combinations of the following values:

Value	Meaning
DT_BOTTOM	Specifies bottom-justified text. This value must be combined (bitwise OR'd) with DT_SINGLELINE.
DT_CALCRECT	Determines the width and height of the rectangle. If there are multiple lines of text, <i>DrawText</i> uses the width of <i>r</i> (the rectangle argument) and extends the base of the rectangle to bound the last line of text. If there is only one line of text, <i>DrawText</i> uses a modified value for the right side of <i>r</i> so that it bounds the last character in the line. In both cases, <i>DrawText</i> returns the height of the formatted text but does not draw the text.
DT_CENTER	Centers text horizontally.
DT_EXPANDTABS	Expands tab characters. The default number of characters per tab is eight.
DT_EXTERNALLEADING	Includes the font external leading in line height. Normally, external leading is not included in the height of a line of text.
DT_LEFT	Aligns text flush-left.
DT_NOCLIP	Draws without clipping. <i>DrawText</i> is somewhat faster when DT_NOCLIP is used.
DT_NOPREFIX	Turns off processing of prefix characters. Normally, <i>DrawText</i> interprets the prefix character & as a directive to underscore the character that follows, and the prefix characters && as a directive to print a single & . By specifying DT_NOPREFIX, this processing is turned off.
DT_RIGHT	Aligns text flush-right.
DT_SINGLELINE	Specifies single line only. Carriage returns and linefeeds do not break the line.
DT_TABSTOP	Sets tab stops. Bits 15-8 (the high-order byte of the low-order word) of the <i>format</i> argument are the number of characters for each tab. The default number of characters per tab is eight.
DT_TOP	Specifies top-justified text (single line only).
DT_VCENTER	Specifies vertically centered text (single line only).

DT_WORDBREAK Specifies word breaking. Lines are automatically broken between words if a word would extend past the edge of the rectangle specified by *r*. A carriage return/line sequence will also break the line.

Note that the `DT_CALCRECT`, `DT_EXTERNALLEADING`, `DT_INTERNAL`, `DT_NOCLIP`, and `DT_NOPREFIX` values cannot be used with the `DT_TABSTOP` value.

DrawText uses this DC's currently selected font, text color, and background color to draw the text. Unless the `DT_NOCLIP` format is used, *DrawText* clips the text so that it does not appear outside the given rectangle. All formatting is assumed to have multiple lines unless the `DT_SINGLELINE` format is given.

If the selected font is too large for the specified rectangle, *DrawText* does not attempt to substitute a smaller font.

If successful, *DrawText* returns the height of the text; otherwise, it returns 0.

See also: `::DrawText`, `TDC::GrayString`, `TDC::TabbedTextOut`, `TDC::TextOut`

Ellipse

```
inline BOOL Ellipse(int x1, int y1, int x2, int y2);
inline BOOL Ellipse(const TPoint& p1, const TPoint& p2);
inline BOOL Ellipse(const TPoint& point, const TSize& size);
inline BOOL Ellipse(const TRect& rect);
```

Draws and fills an ellipse on this DC using the currently selected pen and brush objects. The center of the ellipse is the center of the bounding rectangle specified either by $(x1, y1)/(x2, y2)$ or by the *rect* argument. *Ellipse* returns TRUE if the call is successful; otherwise, it returns FALSE. The current position is neither used nor altered by this call.

See also: `::Ellipse`, `TDC::Arc`

EndPath

```
inline BOOL EndPath();
```



Closes the path bracket and selects the path it defines into this DC. Returns TRUE if the call is successful; otherwise, returns FALSE.

See also: `::EndPath`, `TDC::BeginPath`, `TDC::CloseFigure`

EnumFontFamilies

```
inline int EnumFontFamilies(const char far* family, FONTENUMPROC proc,
                           void* data) const;
```

Enumerates the fonts available to this DC in the font family specified by *family*. The given application-defined callback *proc* (created with `::MakeProcInstance`) is called for each font in the family or until *proc* returns 0. *data* lets you pass both application-specific data and font data to *proc*. If successful, the call returns the last value returned by *proc*.

See also: `::EnumFontFamilies`, `::EnumFontFamProc`

EnumFonts

```
inline int EnumFonts(const char far* faceName, OLDFONTENUMPROC callback,
                    void* data) const;
```

Enumerates the fonts available on this DC for the given *faceName*. The font type, LOGFONT, and TEXTMETRIC data retrieved for each available font is passed to the user-defined *callback* function together with any optional, user-supplied data placed in the *data* buffer. The *callback* function can process this data in any way desired. Enumeration continues until there are no more fonts or until *callback* returns 0. If *faceName* is 0, *EnumFonts* randomly selects and enumerates one font of each available typeface. *EnumFonts* returns the last value returned by *callback()*. Note that OLDFONTENUMPROC is defined as FONTENUMPROC for Win32 only. FONTENUMPROC is a pointer to a user-defined function; it has the following prototype:

```
int CALLBACK EnumFontsProc(LOGFONT *lpf, TEXTMETRIC *lptm, DWORD dwType, LPARAM
                           lpData);
```

where *dwType* specifies the font type: DEVICE_FONTTYPE, RASTER_FONTTYPE, or TRUETYPE_FONTTYPE.

See also: *::EnumFonts*, *::EnumFontsProc*, *TDC::EnumFontFamilies*

EnumMetaFile

```
inline int EnumMetaFile(const TMetaFilePict& metaFile,
                       MFENUMPROC callback, void* data) const;
```

Enumerates the GDI calls within the given *metaFile*. Each such call is retrieved and passed to the given *callback* function, together with any client data from *data*, until all calls have been processed or a callback function returns 0.

See also: *::EnumMetaFile*, *TDC::PlayMetaFile*

EnumObjects

```
int EnumObjects() const;
```

Enumerates the pen or brush objects available for this DC.

ExcludeClipRect

```
inline int ExcludeClipRect(const TRect& rect);
```

Creates a new clipping region for this DC. This new region consists of the current clipping region minus the given rectangle, *rect*. The return value indicates the new clipping region's type as follows:

Region	Meaning
COMPLEXREGION	Clipping Region has overlapping borders.
ERROR	Invalid DC.
NULLREGION	Clipping region is empty.
SIMPLEREGION	Clipping region has no overlapping borders.

See also: `::ExcludeClipRect`, `TDC::GetClipBox`

ExcludeUpdateRgn

```
inline int ExcludeUpdateRgn(HWND wnd);
```

Prevents drawing within invalid areas of a window by excluding an updated region of this DC's window from its clipping region. The return value indicates the resulting clipping region's type as follows:

Region	Meaning
COMPLEXREGION	Clipping Region has overlapping borders.
ERROR	Invalid DC.
NULLREGION	Clipping region is empty.
SIMPLEREGION	Clipping region has no overlapping borders.

See also: `TDC::IntersectClipRect`, `::ExcludeUpdateRgn`, `TDC::GetClipBox`

ExtFloodFill

```
inline BOOL ExtFloodFill(const TPoint& point, TColor color,
                        WORD fillType);
```

Fills an area on this DC starting at *point* and using the currently selected brush object. The *color* argument specifies the color of the boundary or of the region to be filled. The *fillType* argument specifies the type of fill, as follows:

FLOODFILLBORDER	The fill region is bounded by the given <i>color</i> . This style coincides with the filling method used by <code>FloodFill()</code> .
FLOODFILLSURFACE	The fill region is defined by the given <i>color</i> . Filling continues outward in all directions as long as this color is encountered. Use this style when filling regions with multicolored borders.

Not every device supports `ExtFloodFill`, so applications should test first with `TDC::GetDeviceCaps`.

`ExtFloodFill` returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: `TDC::FloodFill`, `::ExtFloodFill`, `TDC::GetDeviceCaps`

ExtTextOut

```
inline virtual BOOL ExtTextOut(int x, int y, WORD options, const TRect* r,
                              const char far* string, int count,
                              const int* dx = 0);
inline BOOL ExtTextOut(const TPoint& p, WORD options, const TRect* r,
                      const char far* string, int count,
                      const int* dx = 0);
```

Draws up to *count* characters of the given null-terminated *string* in the current font on this DC. If *count* is -1, the whole string is written.

An optional rectangle *r* can be specified for clipping, opaquing, or both, as determined by the *options* value. If *options* is set to `ETO_CLIPPED`, the rectangle is used for clipping the drawn text. If *options* is set to `ETO_OPAQUE`, the current background color is used to fill the rectangle. Both options can be used if `ETO_CLIPPED` is OR'd with `ETO_OPAQUE`.

The *(x, y)* or *p* arguments specify the logical coordinates of the reference point that is used to align the first character. The current text-alignment mode can be inspected with `TDC::GetTextAlign` and changed with `TDC::SetTextAlign`. By default, the current position is neither used nor altered by `ExtTextOut`. However, if the align mode is set to `TA_UPDATECP`, `ExtTextOut` ignores the reference point argument(s) and uses or updates the current position as the reference point.

The *dx* argument is an optional array of values used to set the distances between the origins (upper left corners) of adjacent character cells. For example, *dx[i]* represents the number of logical units separating the origins of character cells *i* and *i+1*. If *dx* is 0, `ExtTextOut` uses the default inter-character spacings.

`ExtTextOut` returns `TRUE` if the call is successful; otherwise, it returns `FALSE`.

See also: `TDC::TextOut`, `TDC::GetTextAlign`, `TDC::TabbedTextOut`, `::ExtTextOut`

FillPath



```
inline BOOL FillPath();
```

Closes any open figures in the current path of this DC and fills the path's interior using the current brush and polygon fill mode. After filling the interior, `FillPath` discards the path from this DC.

`FillPath` returns `TRUE` if the call is successful; otherwise, it returns `FALSE`.

See also: `::FillPath`, `TDC::BeginPath`, `TDC::CloseFigure`, `TDC::StrokePath`, `TDC::StrokeAndFillPath`, `TDC::SetPolyFillMode`

FillRect

```
inline BOOL FillRect(int x1, int x2, int y1, int y2, const TBrush& brush);
inline BOOL FillRect(const TRect& rect, const TBrush& brush);
```

Fills the given rectangle on this DC using the specified brush. The fill covers the left and top borders but excludes the right and bottom borders. `FillRect` returns `TRUE` if the call is successful; otherwise, it returns `FALSE`.

See also: `::FillRect`

FillRgn

```
inline BOOL FillRgn(const TRegion& region, const TBrush& brush);
```

Fills the given *region* on this DC using the specified *brush*. `FillRgn` returns `TRUE` if the call is successful; otherwise, it returns `FALSE`.

See also: *TDC::InvertRgn*, *TDC::PaintRgn*, *::FillRgn*

FlattenPath



```
inline BOOL FlattenPath();
```

Transforms any curves in the currently selected path of this DC. All such curves are changed to sequences of linear segments. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *::FlattenPath*, *TDC::WidenPath*, *TDC::BeginPath*

FloodFill

```
inline BOOL FloodFill(const TPoint& point, TColor color);
```

Fills an area on this DC starting at *point* and using the currently selected brush object. The *color* argument specifies the color of the boundary or of the area to be filled. Returns TRUE if the call is successful; otherwise, returns FALSE. *FloodFill* is maintained in the WIN32 API for compatibility with earlier APIs. New WIN32 applications should use *TDC::ExtFloodFill()*.

See also: *TDC::ExtFloodFill*, *::FloodFill*

FrameRect

```
inline BOOL FrameRect(int x1, int x2, int y1, int y2,
                      const TBrush& brush);
```

```
inline BOOL FrameRect(const TRect& rect, const TBrush& brush);
```

Draws a border on this DC around the given rectangle using the given brush. The height and width of the border is one logical unit. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *::FrameRect*

FrameRgn

```
inline BOOL FrameRgn(const TRegion& region, const TBrush& brush,
                    const TPoint& p);
```

Draws a border on this DC around the given region, *region*, using the given brush, *brush*. The width and height of the border is specified by the *p* argument. Returns TRUE if the call is successful; otherwise, returns FALSE.

See also: *::FrameRgn*

GetAspectRatioFilter

```
inline BOOL GetAspectRatioFilter(TSize& size) const;
```

Retrieves the setting of the current aspect-ratio filter for this DC.

See also: *::GetAspectRatioFilter*, *::SetMapperFlags*

GetBkColor

```
inline TColor GetBkColor() const;
```

Returns the current background color of this DC.

See also: *TDC::SetBkColor*, *::GetBkColor*

GetBkMode

```
inline int GetBkMode() const;
```

Returns the background mode of this DC, either OPAQUE or TRANSPARENT.

See also: *TDC::SetBkMode*, *::GetBkMode*

GetBoundsRect

```
inline BOOL GetBoundsRect(TRect& bounds, WORD flags) const;
```

Reports in *bounds* the current accumulated bounding rectangle of this DC or of the Windows manager, depending on the value of *flags*. Returns TRUE if the call is successful; otherwise returns FALSE.

The *flags* argument can be DCB_RESET or DCB_WINDOWMGR or both. The *flags* value work as follows:

DCB_RESET	Forces the bounding rectangle to be cleared after being set in <i>bounds</i> .
DCB_WINDOWMGR	Reports the Windows current bounding rectangle rather than that of this DC.

There are two bounding-rectangle accumulations, one for Windows and one for the application. *GetBoundsRect* returns screen coordinates for the Windows bounds, and logical units for the application bounds. The Windows accumulated bounds can be queried by an application but not altered. The application can both query and alter the DC's accumulated bounds.

See also: *TDC::SetBoundsRect*, *::GetBoundsRect*

GetBrushOrg

```
inline BOOL GetBrushOrg(TPoint& point) const;
```

Places in *point* the current brush origin of this DC. Returns TRUE if successful; otherwise returns FALSE.

See also: *TDC::SetBrushOrgEx*

GetCharABCWidths

```
inline BOOL GetCharABCWidths(UINT firstChar, UINT lastChar, ABC* abc);
```

Retrieves the widths of consecutive characters in the range *firstChar* to *lastChar* from the current TrueType font of this DC. The widths are reported in the array *abc* of ABC structures. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *::GetCharABCWidths*, ABC struct, *TDC::GetCharWidth*

GetCharWidth

```
inline BOOL GetCharWidth(UINT firstChar, UINT lastChar, int* buffer);
```

Retrieves the widths in logical units for a consecutive sequence of characters in the current font for this DC. The sequence is specified by the inclusive range, *firstChar* to *lastChar*, and the widths are copied to the given *buffer*. If a character in the range is not represented in the current font, the

width of the default character is assigned. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: `::GetCharWidth`, `TDC::GetCharABCWidths`

GetClipBox

```
inline int GetClipBox(TRect& rect) const;
inline TRect GetClipBox() const;
```

Places the current clip box size of this DC in *rect*. The clip box is defined as the smallest rectangle bounding the current clipping boundary. The return value indicates the clipping region's type as follows:

Region	Meaning
COMPLEXREGION	Clipping Region has overlapping borders.
ERROR	Invalid DC.
NULLREGION	Clipping region is empty.
SIMPLEREGION	Clipping region has no overlapping borders.

See also: `::GetClipBox`, `TDC::ExcludeClipRect`

GetClipRgn

```
inline BOOL GetClipRgn(TRegion& region) const;
```

Retrieves this DC's current clip-region and, if successful, places a *copy* of it in the *region* argument. You can alter this copy without affecting the current clip-region. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: `::GetClipRgn`

GetCurrentObject

```
inline HANDLE GetCurrentObject(UINT objectType) const;
```



Returns a handle to the currently selected object of the given *objectType* associated with this DC. Returns 0 if the call fails. *objectType* can be OBJ_PEN, OBJ_BRUSH, OBJ_PAL, OBJ_FONT, or OBJ_BITMAP.

See also: `::GetObject`, `::SelectObject`

GetCurrentPosition

```
inline BOOL GetCurrentPosition(TPoint& point) const;
```

Reports in *point* the logical coordinates of this DC's current position. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: `::GetCurrentPosition`

GetDCOrg

```
inline BOOL GetDCOrg(TPoint& point) const;
```

Obtains the final translation origin for this device context and places the value in *point*. This value specifies the offset used to translate device coordinates to client coordinates for points in an application window. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: `::GetDCOrg`

GetDeviceCaps

```
inline virtual int GetDeviceCaps(int index) const;
```

Used under WIN3.1 or later, `GetDeviceCaps` returns capability information about this DC. The *index* argument specifies the type of information required.

See also: `::GetDeviceCaps`

GetDIBits

```
inline BOOL GetDIBits(const TBitmap& bitmap, WORD startScan,
                    WORD numScans, void *HUGE* bits,
                    const BITMAPINFO far& info, WORD usage);
inline BOOL GetDIBits(const TBitmap& bitmap, TDib& dib);
```

The first version retrieves some or all of the bits from the given *bitmap* on this DC and copies them to the *bits* buffer using the DIB (device-independent bitmap) format specified by the BITMAPINFO argument, *info*. *numScan* scanlines of the bitmap are retrieved, starting at scanline *startScan*. The *usage* argument determines the format of the *bmiColors* member of the BITMAPINFO structure, according to the following table:

Value	Meaning
DIB_PAL_COLORS	The color table is an array of 16-bit indices into the current logical palette.
DIB_RGB_COLORS	The color table contains literal RGB values.
DIB_PAL_INDICES	There is no color table for this bitmap. The DIB bits consist of indices into the system palette. No color translation occurs. Only the BITMAPINFOHEADER portion of BITMAPINFO is filled in.

In the second version of `GetDIBits`, the bits are retrieved from *bitmap* and placed in the *dib.Bits* data member of the given *TDib* argument. The BITMAPINFO argument is supplied from *dib.info*.

`GetDIBits` returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: `TDC::SetDIBits`, `::GetDIBits`, `TDib::info`

GetFontData

```
inline DWORD GetFontData(DWORD table, DWORD offset, void* buffer,
                        long data);
```

Retrieves font-metric information from a scalable TrueType font file (specified by *table* and starting at *offset* into this table) and places it in the given *buffer*. *data* specifies the size in bytes of the data to be retrieved. If the call is successful, it returns the number of bytes set in *buffer*; otherwise, -1 is returned.

See also: `::GetFontData`

GetKerningPairs

```
inline int GetKerningPairs(int pairs, KERNINGPAIR far* krnPair);
```

Retrieves up to *pairs* of the kerning pairs for the current font of this DC and copies them into the *krnPair* array of KERNINGPAIR structures. If successful, the function returns the actual number of pairs retrieved. If the font has more than *pairs* kerning pairs, the call fails and returns 0. The *krnPair* array must allow for at least *pairs* KERNINGPAIRS structures. If *krnPair* is set to 0, GetKerningPairs returns the total number of kerning pairs for the current font.

See also: `::GetKerningPairs, KERNINGPAIR` struct

GetMapMode

```
inline int GetMapMode() const;
```

If successful, *GetMapMode* returns the current window mapping mode of this DC; otherwise, it returns 0. The mapping mode defines how logical coordinates are mapped to device coordinates. It also controls the orientation of the device's x- and y-axes. The mode values are shown in the following table:

Value	Meaning
MM_ANISOTROPIC	Logical units are mapped to arbitrary units with arbitrarily scaled axes. The <i>SetWindowExtEx</i> and <i>SetViewportExtEx</i> functions must be used to specify the desired units, orientation, and scaling.
MM_HIENGLISH	Each logical unit is mapped to 0.001 inch. Positive x is to the right; positive y is at the top.
MM_HIMETRIC	Each logical unit is mapped to 0.01 millimeter. Positive x is to the right; positive y is at the top.
MM_ISOTROPIC	Logical units are mapped to arbitrary units with equally scaled axes; that is, one unit along the x-axis is equal to one unit along the y-axis. The <i>SetWindowExtEx</i> and <i>SetViewportExtEx</i> functions must be used to specify the desired units and the orientation of the axes. GDI makes adjustments as necessary to ensure that the x and y units remain the same size (e.g., if you set the window extent, the viewport is adjusted to keep the units isotropic).
MM_LOENGLISH	Each logical unit is mapped to 0.01 inch. Positive x is to the right; positive y is at the top.
MM_LOMETRIC	Each logical unit is mapped to 0.1 millimeter. Positive x is to the right; positive y is at the top.
MM_TEXT	Each logical unit is mapped to one device pixel. Positive x is to the right; positive y is at the bottom.
MM_TWIPS	Each logical unit is mapped to one twentieth of a printer's point (1/1440 inch). Positive x is to the right; positive y is at the top.

See also: *TDC::SetMapMode*, *::GetMapMode*

GetNearestColor

```
inline TColor GetNearestColor(TColor Color) const;
```

Returns the nearest color to the given *Color* argument for the current palette of this DC.

See also: *::GetNearestColor*

GetOutlineTextMetrics

```
inline DWORD GetOutlineTextMetrics(UINT data,
                                   OUTLINETEXMETRIC far& otm);
inline WORD GetOutlineTextMetrics(UINT data,
                                   OUTLINETEXMETRIC far& otm);
```



Retrieves metric information for TrueType fonts on this DC and copies it to the given array of *OUTLINETEXMETRIC* structures, *otm*. This structure contains a *TEXTMETRIC* and several other metric members, as well as four string-pointer members for holding family, face, style, and full font names. Since memory must be allocated for these variable-length strings in addition to the font metric data, you must pass (via the *data* argument) the total number of bytes required for the retrieved data. If *GetOutlineTextMetrics* is called with *otm = 0*, the function returns the total buffer size required. You can then assign this value to *data* in subsequent calls.

The WIN32 version returns a *DWORD* value.

Returns nonzero if the call is successful; otherwise, returns 0.

See also: *::GetOutlineTextMetrics*, *OUTLINETEXMETRICS* struct, *TDC::GetTextMetrics*

GetPixel

```
inline TColor GetPixel(int x, int y) const;
inline TColor GetPixel(const TPoint& point) const;
```

Returns the color of the pixel at the given location.

See also: *TDC::SetPixel*, *::GetPixel*

GetPolyFillMode

```
inline int GetPolyFillMode() const;
```

Returns the current polygon-filling mode for this DC, either *ALTERNATE* or *WINDING*.

See also: *TDC::SetPolyFillMode*, *::GetPolyFillMode*

GetROP2()

```
inline int GetROP2() const;
```

Returns the current drawing (raster operation) mode of this DC.

See also: *TDC::SetROP2()*, *::GetROP2*

GetStretchBltMode

```
inline int GetStretchBltMode() const;
```

Returns the current stretching mode for this DC: BLACKONWHITE, COLORONCOLOR, or WHITEONBLACK. The stretching mode determines how bitmaps are stretched or compressed by the *StretchBlt* function.

See also: *TDC::SetStretchBltMode*, *::GetStretchBltMode*, *TDC::StretchBlt*

GetSystemPaletteEntries inline UINT GetSystemPaletteEntries(int start, int num, PALETTEENTRY far* entries) const;

Retrieves a range of up to *num* palette entries, starting at *start*, from the system palette to the *entries* array of PALETTEENTRY structures. Returns the actual number of entries transferred.

See also: *::GetSystemPaletteEntries*

GetSystemPaletteUse inline UINT GetSystemPaletteUse() const;

Determines whether this DC has access to the full system palette. Returns SYSPAL_NOSTATIC or SYSPAL_STATIC.

See also: *TDC::SetSystemPaletteUse*, *::GetSystemPaletteUse*

GetTabbedTextExtent inline BOOL GetTabbedTextExtent(const char far* string, int stringLen, int numPositions, const int* positions, TSize& size) const;
inline TSize GetTabbedTextExtent(const char far* string, int stringLen, int numPositions, const int* positions) const;

Computes the extent (width and height) in logical units of the text line consisting of *stringLen* characters from the null-terminated *string*. The extent is calculated from the metrics of the current font or this DC, but ignores the current clipping region. In the first version of *GetTabbedTextExtent*, the extent is returned in *size*; in the second version, the extent is the returned *TSize* object. Width is *size.x* and height is *size.y*.

The width calculation includes the spaces implied by any tab codes in the string. Such tab codes are interpreted using the *numPositions* and *positions* arguments. The *positions* array specifies *numPositions* tab stops given in device units. The tab stops must have strictly increasing values in the array. If *numPositions* and *positions* are both 0, tabs are expanded to eight times the average character width. If *numPositions* is 1, all tab stops are taken to be *positions[0]* apart.

If kerning is being applied, the sum of the extents of the characters in a string might not equal the extent of the string.

See also: *::GetTabbedTextExtent*, *TDC::TabbedTextOut*, *TDC::GetTextExtent*

GetTextAlign

```
inline UINT GetTextAlign() const;
```

If successful, *GetTextAlign* returns the current text-alignment flags for this DC; otherwise, it returns the value `GDI_ERROR`. The text-alignment flags determine how *TDC::TextOut()* and *TDC::ExtTextOut* align text strings in relation to the first character's screen position. *GetTextAlign* returns certain combinations of the flags listed in the following table:

Value	Meaning
TA_BASELINE	The reference point will be on the baseline of the text.
TA_BOTTOM	The reference point will be on the bottom edge of the bounding rectangle.
TA_TOP	The reference point will be on the top edge of the bounding rectangle.
TA_CENTER	The reference point will be aligned horizontally with the center of the bounding rectangle.
TA_LEFT	The reference point will be on the left edge of the bounding rectangle.
TA_RIGHT	The reference point will be on the right edge of the bounding rectangle.
TA_NOUPDATECP	The current position is not updated after each text output call.
TA_UPDATECP	The current position is updated after each text output call.
	When the current font has a vertical default baseline (as with Kanji) the following values replace TA_BASELINE and TA_CENTER:
VTA_BASELINE	The reference point will be on the baseline of the text.
VTA_CENTER	The reference point will be aligned vertically with the center of the bounding rectangle.

The text-alignment flags are not necessarily single bit-flags and might be equal to 0. The flags must be examined in groups of the following related flags:

- TA_LEFT, TA_RIGHT, and TA_CENTER
- TA_BOTTOM, TA_TOP, and TA_BASELINE
- TA_NOUPDATECP and TA_UPDATECP

If the current font has a vertical default baseline (as with Kanji), these are groups of related flags:

- TA_LEFT, TA_RIGHT, and VTA_BASELINE
- TA_BOTTOM, TA_TOP, and VTA_CENTER
- TA_NOUPDATECP and TA_UPDATECP

To verify that a particular flag is set in the return value of this function, the application must perform the following steps:

1. Apply the bitwise OR operator to the flag and its related flags.
2. Apply the bitwise AND operator to the result and the return value.
3. Test for the equality of this result and the flag.

The following example shows a method for determining which horizontal alignment flag is set:

```
switch ((TA_LEFT | TA_RIGHT | TA_CENTER) & dc.GetTextAlign()) {
    case TA_LEFT:
        ...
    case TA_RIGHT:
        ...
    case TA_CENTER:
        ...
}
```

See also: *TDC::SetTextAlign*, *::GetTextAlign*, *TDC::TextOut*, *TDC::ExtTextOut*

GetTextColor

```
inline TColor GetTextColor() const;
```

Returns the current text color of this DC. The text color determines the color displayed by *TDC::TextOut()* and *TDC::ExtTextOut()*.

See also: *TDC::SetTextColor*, *::GetTextColor*, *TDC::TextOut*, *TDC::ExtTextOut*

GetTextExtent

```
inline BOOL GetTextExtent(const char far* string, int stringLen,
                          TSize& size);
inline TSize GetTextExtent(const char far* string, int stringLen);
```

Computes the extent (width and height) in logical units of the text line consisting of *stringLen* characters from the null-terminated *string*. The extent is calculated from the metrics of the current font or this DC, but ignores the current clipping region. In the first version of *GetTextExtent* the extent is returned in *size*; in the second version, the extent is the returned *TSize* object. Width is *size.x* and height is *size.y*.

If kerning is being applied, the sum of the extents of the characters in a string might not equal the extent of the string.

GetTextExtent is not supported under WIN32. WIN32 applications should use *TDC::GetTextExtentPoint*.

See also: *::GetTextExtent*, *TDC::GetTextExtentPoint*

GetTextFace

```
inline int GetTextFace(int count, char far* facename) const;
```

Retrieves the typeface name for the current font on this DC. Up to *count* characters of this name are copied to *facename*. If successful, *GetTextFace* returns the number of characters actually copied; otherwise, it returns 0.

See also: *::GetTextFace*, *TDC::GetTextAlign*, *TDC::GetTextMetrics*

GetTextMetrics

```
inline BOOL GetTextMetrics(TEXTMETRIC far& metrics) const;
```

Fills the *metrics* structure with metrics data for the current font on this DC. Returns TRUE if the call is successful; otherwise, returns FALSE.

See also: *::GetTextMetrics*, *TEXTMETRIC* struct

```
DWORD GetGlyphOutline(UINT chr, UINT format, GLYPHMETRICS far& gm,
                      DWORD buffSize, void far* buffer,
                      const MAT2 far& mat2);
```

Retrieves TrueType metric and other data for the given character, *chr*, on this DC and places it in *gm* and *buffer*. The *format* argument specifies the format of the retrieved data as follows:

Value	Meaning
1	Retrieves the glyph bitmap.
2	Retrieves the curve data points in the rasterizer's native format and uses the font's design units. With this value of <i>format</i> , the <i>mat2</i> transformation argument is ignored.

The *gm* argument specifies the *GLYPHMETRICS* structure that describes the placement of the glyph in the character cell. *buffSize* specifies the size of *buffer* that receives data about the outline character. If either *buffSize* or *buffer* are 0, *GetGlyphOutline* returns the required buffer size. Applications can rotate characters retrieved in bitmap format (*format* = 1) by specifying a 2 x 2 transformation matrix via the *mat2* argument.

GetGlyphOutline returns a positive number if the call is successful; otherwise, it returns *GDI_ERROR*.

See also: *::GetGlyphOutline*, *GLYPHMETRICS* struct, *TDC::GetOutlineTextMetrics*

GetViewportExt

```
inline BOOL GetViewportExt(TSize& extent) const;
inline TSize GetViewportExt() const;
```

The first version retrieves this DC's current viewport's x- and y-extents (in device units) and places the values in *extent*. This version returns TRUE if the call is successful; otherwise, it returns FALSE. The second version returns only these x- and y-extents.

The *extent* value determines the amount of stretching or compression needed in the logical coordinate system to fit the device coordinate system.

extent also determines the relative orientation of the two coordinate systems.

See also: *TDC::SetViewportExt*, *::GetViewportExt*

GetViewportOrg

```
inline BOOL GetViewportOrg(TPoint& point) const;
inline TPoint GetViewportOrg() const;
```

The first version sets in the *point* argument the x- and y-extents (in device-units) of this DC's viewport. It returns TRUE if the call is successful; otherwise, it returns FALSE. The second version returns the x- and y-extents (in device-units) of this DC's viewport.

See also: *TDC::SetViewportOrg*, *TDC::OffsetViewportOrg*, *::GetViewportOrg*

GetWindowExt

```
inline BOOL GetWindowExt(TSize& extent) const;
inline TSize GetWindowExt() const;
```

Retrieves this DC's window current x- and y-extents (in device units). The first version places the values in *extent* and returns TRUE if the call is successful; otherwise, it returns FALSE. The second version returns the current extent values. The *extent* value determines the amount of stretching or compression needed in the logical coordinate system to fit the device coordinate system. *extent* also determines the relative orientation of the two coordinate systems.

See also: *TDC::SetWindowExt*, *::GetWindowExt*

GetWindowOrg

```
inline BOOL GetWindowOrg(TPoint& point) const;
inline TPoint GetWindowOrg() const;
```

Places in *point* the x- and y-coordinates of the origin of the window associated with this DC. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *TDC::SetWindowOrg*, *TDC::OffsetWindowOrg*, *::GetWindowOrg*

GrayString

```
inline virtual BOOL GrayString(const TBrush& brush,
                               GRAYSTRINGPROC outputFunc,
                               const char far* string, int count,
                               const TRect& r);
```

Draws in the given rectangle (*r*) up to *count* characters of gray text from *string* using the given *brush* and the current font for this DC. If *count* is -1 and *string* is null-terminated, the whole string is written. The rectangle must be specified in logical units. If *brush* is 0, the text is grayed with the same brush used to draw window text on this DC. Gray text is primarily used to indicate disabled commands and menu items.

GrayString writes the selected text to a memory bitmap, grays the bitmap, then displays the result. The graying is performed regardless of the current brush and background color.

The *outputFunc* pointer-to-function can specify the procedure-instance of an application-supplied drawing function. If *outputFunc* is 0, *GrayString* uses *TextOut* and *string* is assumed to be a normal, null-terminated character string. If *string* cannot be handled by *TextOut* (for example, if the string is stored as a bitmap), you must provide a suitable drawing function via *outputFunc*.

If the device supports a solid gray color, it is possible to draw gray strings directly without using *GrayString*. Call *GetSysColor* to find the color value; for example, *G* of COLOR_GRAYTEXT. If *G* is nonzero (non-black), you can set the text color with *SetTextColor(G)* and then use any convenient text-drawing function.

GrayString returns TRUE if the call is successful; otherwise, it returns FALSE. Failure can result if *TextOut* or *outputFunc* return FALSE, or if there is insufficient memory to create the bitmap.

See also: *::GrayString*, *TDC::TextOut*

IntersectClipRect

```
int IntersectClipRect(const TRect& rect);
```

Creates a new clipping region for this DC's window by forming the intersection of the current region with the rectangle specified by *rect*. The return value indicates the resulting clipping region's type as follows:

Region	Meaning
COMPLEXREGION	Clipping Region has overlapping borders.
ERROR	Invalid DC.
NULLREGION	Clipping region is empty.
SIMPLEREGION	Clipping region has no overlapping borders.

See also: *TDC::GetClipBox*, *::IntersectClipRect*

InvertRect

```
inline BOOL InvertRect(int x1, int x2, int y1, int y2);
inline BOOL InvertRect(const TRect& rect);
```

Inverts the given rectangle on this DC. On monochrome displays, black and white pixels are interchanged. On color displays, inversion depends on how the colors are generated for particular displays. Calling *InvertRect* an even number of times restores the original colors. *InvertRect* returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: *::InvertRect*

InvertRgn

```
inline BOOL InvertRgn(const TRegion& region);
```

Inverts the given *region*, on this DC. On monochrome displays, black and white pixels are interchanged. On color displays, inversion depends on how the colors are generated for particular displays. Calling *InvertRegion* an even number ($n \geq 2$) of times restores the original colors. Returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: *TDC::PaintRgn*, *TDC::FillRgn*, *::InvertRgn*

LineDDA

```
inline BOOL LineDDA(int x1, int y1, int x2, int y2, LINEDDAPROC proc,
                   LPARAM lParam);
inline BOOL TextRect(const TPoint& p1, const TPoint& p2, LINEDDAPROC proc,
                   LPARAM lParam);
```

Determines which pixels should be highlighted for a line given by the start ($x1, y1$ or point $p1$) and end ($x2, y2$ or point $p2$) point arguments. Each point along the path (excluding the end point) is passed to the user-defined callback function, *proc*, together with the optional user-supplied data via *lParam*. The *proc* argument must be of type pointer to *LineDDAFunc*, a function declared as

```
void CALLBACK LineDDAFunc(int x, int y, LPARAM lParam);
```

where x and y specify the current point in the path.

LineDDA returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: *::LineDDA*

LineTo

```
inline BOOL LineTo(int x, int y);
inline BOOL LineTo(const TPoint& point);
```

Draws a line on this DC using the current pen object. The line is drawn from the current position up to, but not including, the given end point, which is specified by (x, y) or by *point*. If the call is successful, *LineTo* returns TRUE and the current point is reset to *point*; otherwise, it returns FALSE.

See also: *::LineTo*

LPToDP

```
inline BOOL LPToDP(TPoint* points, int count = 1) const;
```

Converts each of the *count* points in the *points* array from logical points to device points. The conversion depends on this DC's current mapping mode and the settings of its window and viewport origins and extents. Returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: *TDC::DPtoLP*, *::LPToDP*

MaskBlt

```
inline BOOL MaskBlt(const TRect& dst, const TDC& srcDC, const TPoint& src,
                  const TBitmap& maskBm, const TPoint& maskPos,
                  DWORD rop);
```

Copies a bitmap from the given source DC to this DC. *MaskBlt* combines the color data from source and destination bitmaps using the given mask and raster operation. The *srcDC* argument specifies the DC from which the source bitmap will be copied. The destination bitmap is given by the rectangle, *dst*. The source bitmap has the same width and height as *dst*. The *src* argument specifies the logical coordinates of the upper left corner of the source bitmap. The *maskBm* argument specifies a monochrome mask bitmap. An error will occur if *maskBm* is not monochrome. The *maskPos* argument gives the upper left corner coordinates of the mask. The raster-operation code, *rop*, specifies how the source, mask, and destination bitmaps combine to produce the new destination bitmap. The raster-operation codes are as follows:

Value of <i>rop</i>	Meaning
BLACKNESS	Fill <i>dst</i> with index-0 color of physical palette (default is black).
DSTINVERT	Invert <i>dst</i> .
MERGECOPY	Merge the colors of source with mask with Boolean AND.
MERGEPAINT	Merge the colors of inverted-source with the colors of <i>dst</i> using Boolean OR.
NOTSRCCOPY	Copy inverted-source to <i>dst</i> .
NOTSRCERASE	Combine the colors of source and <i>dst</i> using Boolean OR, then invert result.
PATCOPY	Copy mask to <i>dst</i> .
PATINVERT	Combine the colors of mask with the colors of <i>dst</i> using Boolean XOR.
PATPAINT	Combine the colors of mask with the colors of inverted-source using Boolean OR, then combine the result with the colors of <i>dst</i> using Boolean OR.
SRCAND	Combine the colors of source and <i>dst</i> using the Boolean AND.
SRCOPY	Copy source directly to <i>dst</i> .
SRCERASE	Combine the inverted colors of <i>dst</i> with the colors of source using Boolean AND.
SRCPAINT	Combine the colors of source and <i>dst</i> using Boolean OR.
WHITENESS	Fill <i>dst</i> with index-1 color of physical palette (default is white).

If *rop* indicates an operation that excludes the source bitmap, the *srcDC* argument must be 0. A value of 1 in the mask indicates that the destination and source pixel colors should be combined using the high-order word of *rop*. A value of 0 in the mask indicates that the destination and source pixel colors should be combined using the low-order word of *rop*. If the mask rectangle is smaller than *dst*, the mask pattern will be suitably duplicated.

When recording an enhanced metafile, an error occurs if the source DC identifies the enhanced metafile DC.

If a rotation or shear transformation is in effect for the source DC when *MaskBlt* is called, an error occurs. Other transformations are allowed. If necessary, *MaskBlt* will adjust the destination and mask color formats to match that of the source bitmaps. Before using *MaskBlt*, an application should call *GetDeviceCaps* to determine if the source and destination DCs support *MaskBlt*.

MaskBlt returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: *::MaskBlt*, *TDC::BitBlt*, *TDC::PlgBlt*, *TDC::GetDeviceCaps*

ModifyWorldTransform inline BOOL ModifyWorldTransform(XFORM far& xform, DWORD mode);



Changes the current world transformation for this DC using the given *xform* and *mode* arguments. *mode* determines how the given XFORM structure is applied, as listed below.

Value	Meaning
MWT_IDENTITY	Resets the current world transformation using the identity matrix. If this mode is specified, the XFORM structure pointed to by <i>lpXform</i> is ignored.
MWT_LEFTMULTIPLY	Multiplies the current transformation by the data in the XFORM structure. (The data in the XFORM structure becomes the left multiplicand, and the data for the current transformation becomes the right multiplicand.)
MWT_RIGHTMULTIPLY	Multiplies the current transformation by the data in the XFORM structure. (The data in the XFORM structure becomes the right multiplicand, and the data for the current transformation becomes the left multiplicand.) <i>ModifyWorldTransform</i> returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: *TDC::SetWorldTransform*, *::ModifyWorldTransform*

MoveTo

```
inline BOOL MoveTo(int x, int y);
inline BOOL MoveTo(const TPoint& point);
inline BOOL MoveTo(const TPoint& point, TPoint& oldPoint);
```

Moves the current position of this DC to the given *x*- and *y*-coordinates or to the given *point*. The third version, corresponding to `::MoveToEx`, sets the previous current position in *oldPoint*. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: `::MoveTo`, `::MoveToEx`

OffsetClipRgn

```
inline int OffsetClipRgn(const TPoint& delta);
```

Moves the clipping region of this DC by the *x*- and *y*-offsets specified in *delta*. The return value indicates the resulting clipping region's type as follows:

Region	Meaning
COMPLEXREGION	Clipping region has overlapping borders.
ERROR	Invalid DC.
NULLREGION	Clipping region is empty.
SIMPLEREGION	Clipping region has no overlapping borders.

See also: `::OffsetClipRgn`, `TDC::GetClipBox`

OffsetViewportOrg

```
inline virtual BOOL OffsetViewportOrg(const TPoint& delta, TPoint* oldOrg = 0);
```

Modifies this DC's viewport origin relative to the current values. The *delta* *x*- and *y*-components are added to the previous origin and the resulting point becomes the new viewport origin. The previous origin is saved in *oldOrg*. Returns TRUE if the call is successful; otherwise, returns FALSE.

See also: `TDC::SetViewportOrg`, `TDC::GetViewportOrg`, `::OffsetViewportOrg`

OffsetWindowOrg

```
inline BOOL OffsetWindowOrg(const TPoint& delta, TPoint* oldOrg = 0);
```

Modifies this DC's window origin relative to the current values. The *delta* *x*- and *y*-components are added to the previous origin and the resulting point becomes the new window origin. The previous origin is saved in *oldOrg*. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: `TDC::GetWindowOrg`, `TDC::SetWindowOrg`, `::OffsetWindowOrg`

operator HDC()

```
operator HDC() const{return Handle;}
```

Typecasting operator. Converts a pointer to type *HDC* (the Windows data type representing the handle to a DC).

PaintRgn

```
inline BOOL PaintRgn(const TRegion& region);
```

Paints (fills) the given *region* on this DC using the currently selected brush. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: `TDC::FillRgn`, `::PaintRgn`, `TDC::SelectObject`

PatBlt

```
inline BOOL PatBlt(int x, int y, int w, int h, DWORD rop);
inline BOOL PatBlt(const TRect& dst, DWORD rop);
```

Paints the given rectangle using the currently selected brush for this DC. The rectangle can be specified by its upper left coordinates (x , y), width w , and height h , or by a single *TRect* argument. The raster-operation code, *rop*, determines how the brush and surface color(s) are combined, as explained in the following table:

Value	Meaning
PATCOPY	Copies pattern to destination bitmap.
PATINVERT	Combines destination bitmap with pattern using the Boolean OR operator.
DSTINVERT	Inverts the destination bitmap.
BLACKNESS	Turns all output to binary 0s.
WHITENESS	Turns all output to binary 1s.

The allowed values of *rop* for this function are a limited subset of the full 256 ternary raster-operation codes; in particular, an operation code that refers to a source cannot be used with *PatBlt*.

Not all devices support the *PatBlt* function, so applications should call *TDC::GetDeviceCaps* to check the features supported by this DC.

PatBlt returns TRUE if the call is successful; otherwise returns FALSE.

See also: *::PatBlt*, *TDC::GetDeviceCaps*

PathToRegion

```
inline HRGN PathToRegion();
```



If successful, *PathToRegion* returns a region created from the closed path in this DC; otherwise, it returns 0.

See also: *::PathToRegion*

Pie

```
inline BOOL Pie(int x1, int y1, int x2, int y2, int x3, int y3, int x4,
               int y4);
inline BOOL Pie(const TRect& rect, const TPoint& start,
               const TPoint& end);
```

Using the currently selected pen and brush objects, draws and fills a pie-shaped wedge by drawing an elliptical arc whose center and end points are joined by lines. The center of the ellipse is the center of the rectangle specified either by $(x1, y1)/(x2, y2)$ or by the *rect* argument. The starting/ending points of pie are specified either by $(x3, y3)/(x4, y4)$ or by the points *Start* and *End*. Returns TRUE if the call is successful; otherwise, returns FALSE. The current position is neither used nor altered by this call.

See also: `::Pie`, `TDC::Chord`, `TDC::Arc`

PlayMetaFile

```
inline BOOL PlayMetaFile(const TMetaFilePict& metaFile);
```

Plays the contents of the given *metaFile* on this DC. The metafile can be played any number of times. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: `::PlayMetaFile`, `TDC::EnumMetaFile`, `TDC::PlayMetaFileRecord`

PlayMetaFileRecord

```
inline void PlayMetaFileRecord(HANDLETABLE far& Handletable,
                               METARECORD far& metaRecord, int count);
```

Plays the metafile record given in *metaRecord* to this DC by executing the GDI function call contained in that record. *Handletable* specifies the object handle table to be used. *count* specifies the number of handles in the table.

See also: `::PlayMetaFileRecord`, `TDC::PlayMetaFile`, `TDC::EnumMetaFile`

PlgBlt

```
inline BOOL PlgBlt(const TPoint& dst, const TDC& srcDC, const TRect& src,
                  const TBitmap& maskBm, const TPoint& maskPos,
                  DWORD rop);
```



Performs a bit-block transfer from the given source DC to this DC. Color bits are copied from the *src* rectangle on *srcDC*, the source DC, to the parallelogram *dst* on this DC. The *dst* array specifies three points A, B, and C as the corners of the destination parallelogram. The fourth point D is generated internally from the vector equation $D = B + C - A$. The upper left corner of *src* is mapped to A, the upper right corner to B, the lower left corner to C, and the lower right corner to D. An optional monochrome bitmap can be specified by the *maskBm* argument. (If *maskBm* specifies a valid monochrome bitmap, *PlgBlt* uses it to mask the color bits in the source rectangle. An error occurs if *maskBm* is not a monochrome bitmap.) *maskPos* specifies the upper left corner coordinates of the mask bitmap. With a valid *maskBm*, a value of 1 in the mask causes the source color pixel to be copied to *dst*; a value of 0 in the mask indicates that the corresponding color pixel in *dst* will not be changed. If the mask rectangle is smaller than *dst*, the mask pattern will be suitably duplicated.

The destination coordinates are transformed according to this DC (the destination DC). The source coordinates are transformed according to the source DC. If a rotation or shear transformation is in effect for the source DC when *PlgBlt* is called, an error occurs. Other transformations, such as scaling, translation, and reflection are allowed. The stretching mode of this DC (the destination DC) determines how *PlgBlt* will stretch or compress the pixels if necessary. When recording an enhanced metafile, an error occurs if the source DC identifies the enhanced metafile DC.

If necessary, *PlgBlt* adjusts the source color formats to match that of the destination. An error occurs if the source and destination DCs are incompatible. Before using *PlgBlt*, an application should call *GetDeviceCaps* to determine if the source and destination DCs are compatible.

PlgBlt returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: *::PlgBlt*, *TDC::GetDeviceCaps*, *TDC::SetStretchBltMode*, *MaskBlt*, *TDC::StretchBlt*

PolyBezier

```
inline BOOL PolyBezier(const TPoint* points, int count);
```



Draws one or more connected cubic Bezier splines through the points specified in the *points* array using the currently selected pen object. The first spline is drawn from the first to the fourth point of the array using the second and third points as controls. Subsequent splines, if any, each require three additional points in the array, since the previous end point is taken as the next spline's start point. The *count* argument (≥ 4) specifies the total number of points needed to specify the complete drawing. To draw *n* splines, *count* must be set to $(3n + 1)$. Returns TRUE if the call is successful; otherwise returns FALSE. The current position is neither used nor altered by this call. The resulting figure is not filled.

See also: *TDC::PolyBezierTo*, *::PolyBezier*

PolyBezierTo

```
inline BOOL PolyBezierTo(const TPoint* points, int count);
```



Draws one or more connected cubic Bezier splines through the points specified in the *points* array using the currently selected pen object. The first spline is drawn from the current position to the third point of the array using the first and second points as controls. Subsequent splines, if any, each require three additional points in the array, since the previous end point is taken as the next spline's start point. The *count* argument (≥ 4) specifies the total number of points needed to specify the complete drawing. To draw *n* splines, *count* must be set to $3n$. Returns TRUE if the call is successful; otherwise returns FALSE. The current position is moved to the end point of the final Bezier curve. The resulting figure is not filled.

See also: *TDC::PolyBezier*, *::PolyBezierTo*

PolyDraw

```
inline BOOL PolyDraw(const TPoint* points, BYTE* types, int count);
```



Draws one or more, possibly disjoint, sets of line segments and/or Bezier splines on this DC using the currently selected pen object. The *count* points in the *points* array provide the end points for each line segment and/or the end points and control points for each Bezier spline. The *count* BYTES in the *types* array determine how the corresponding point in *points* is to be interpreted:

Byte	Meaning
PT_BEZIERTO	This point is a control or end point for a Bezier spline. PT_BEZIERTO types must appear in sets of three: the current position is the Bezier start point; the first two PT_BEZIERTO points are the Bezier control points; and the third PT_BEZIERTO point is the Bezier end point, which becomes the new current point. An error occurs if the PT_BEZIERTO types do not appear in sets of three. An end-point PT_BEZIERTO can be bit-wise OR'd with PT_CLOSEFIGURE to indicate that the current figure is to be closed by drawing a spline from this end point to the start point of the most recent disjoint figure.
PT_CLOSEFIGURE	Optional flag that can be bit-wise OR'd with PT_LINETO or PT_BEZIERTO, as explained above. Closure updates the current point to the new end point.
PT_LINETO	A line is drawn from the current position to this point, which then becomes the new current point. PT_LINETO can be bit-wise OR'd with PT_CLOSEFIGURE to indicate that the current figure is to be closed by drawing a line segment from this point to the start point of the most recent disjoint figure.
PT_MOVETO	This point starts a new (disjoint) figure and becomes the new current point.

PolyDraw is an alternative to consecutive calls to *MoveTo*, *LineTo*, *Polyline*, *PolyBezier*, and *PolyBezierTo*. If there is an active path invoked via *BeginPath*, *PolyDraw* will add to this path.

Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *::PolyDraw*, *TDC::MoveTo*, *TDC::LineTo*, *TDC::PolyBezierTo*, *PolyBezier*, *TDC::Polyline*, *TDC::BeginPath*

Polygon

```
inline BOOL Polygon(const TPoint* points, int count);
```

Draws and fills a closed polygon of *count* (≥ 2) line segments on this DC using the current pen and polygon-filling mode. The *points* array specifies the vertices of the polygon to be drawn. The polygon is automatically closed, if necessary, by drawing a line from the last to the first vertex. The current position is neither used nor altered by *Polygon*. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *TDC::Polyline*, *::Polygon*, *TDC::SetPolyFillMode*, *TDC::GetPolyFillMode*

Polyline

```
inline BOOL Polyline(const TPoint* points, int count);
```

Draws a sequence of *count* (≥ 2) line segments on this DC using the current pen object. The *points* array specifies the sequence of points to be

connected. The current position is neither used nor altered by *Polyline*. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *TDC::Polygon*, *::Polyline*, *TDC::PolyPolyline*

PolylineTo



```
inline BOOL PolylineTo(const TPoint* points, int count);
```

Draws one or more connected line segments on this DC using the currently selected pen object. The first line is drawn from the current position to the first of the *count* points in the *points* array. Subsequent lines, if any, connect the remaining points in the array, with each end point providing the start point of the next segment. The final end point becomes the new current point. No filling occurs even if a closed figure is drawn. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *TDC::PolyDraw*, *::PolylineTo*, *TDC::LineTo*

PolyPolygon

```
inline BOOL PolyPolygon(const TPoint* points, const int* PolyCounts,
                        int count);
```

Draws and fills a series of *count* (≥ 2), possibly overlapping, closed polygons on this DC using the current pen and polygon-filling mode. The *points* array specifies the vertices of the polygons to be drawn. *PolyCounts* is an array of *count* integers specifying the number of vertices in each polygon. Each polygon must be a closed polygon. The current position is neither used nor altered by *Polygon*. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *::PolyPolygon*, *TDC::PolyPolyline*, *TDC::SetPolyFillMode*, *TDC::GetPolyFillMode*

PolyPolyline



```
inline BOOL PolyPolyline(const TPoint* points, const int* PolyCounts,
                          int count);
```

Draws a series of *count* polylines (connected line segments) on this DC using the currently selected pen object. The resulting figures are not filled. The *PolyCounts* array provides *count* integers specifying the number of points (≥ 2) in each polyline. The *points* array provides, consecutively, each of the points to be connected. Returns TRUE if the call is successful; otherwise returns FALSE. The current position is neither used nor altered by this call.

See also: *::PolyPolyline*, *TDC::Polyline*, *TDC::PolyPolygon*

PtVisible

```
inline BOOL PtVisible(const TPoint& point) const;
```

Returns TRUE if the given *point* lies within the clipping region of this DC; otherwise returns FALSE.

See also: *TDC::RectVisible*, *::PtVisible*

- RealizePalette** `inline int RealizePalette();`
- Maps to the system palette the logical palette entries selected into this DC.
- See also: `::RealizePalette`
- Rectangle** `inline BOOL Rectangle(int x1, int y1, int x2, int y2);`
`inline BOOL Rectangle(const TPoint& p1, const TPoint& p2);`
`inline BOOL Rectangle(const TPoint& point, const TSize& s);`
`inline BOOL Rectangle(const TRect& rect);`
- Draws and fills a rectangle of the given size on this DC with the current pen and brush objects. The current position is neither used nor altered by this call. Returns TRUE if the call is successful; otherwise returns FALSE.
- See also: `TDC::RoundRect`, `::Rectangle`
- RectVisible** `inline BOOL RectVisible(const TRect& rect) const;`
- Returns TRUE if any part of the given *rectangle* lies within the clipping region of this DC; otherwise returns FALSE.
- See also: `TDC::PtVisible`, `::RectVisible`
- ResetDC** `inline virtual BOOL ResetDC(DEVMODE far& devMode);`
- Updates this DC using data in the given *devMode* structure. Returns TRUE if the call is successful; otherwise returns FALSE.
- See also: `::ResetDC`
- RestoreBrush** `void RestoreBrush();`
- Restores the original GDI brush object to this DC.
- See also: `::SelectObject`, `TDC::OrgBrush`
- RestoreDC** `inline virtual BOOL RestoreDC(int savedDC = -1);`
- Restores the given *savedDC*. Returns TRUE if the context is successfully restored; otherwise returns FALSE.
- See also: `TDC::SaveDC`, `::RestoreDC`
- RestoreFont** `virtual void RestoreFont();`
- Restores the original GDI font object to this DC.
- See also: `::SelectObject`, `TDC::OrgFont`
- RestoreObjects** `void RestoreObjects();`
- Restores all the original GDI objects to this DC.

See also: *::SelectObject*

RestorePalette

```
void RestorePalette();
```

Restores the original GDI palette object to this DC.

See also: *::SelectPalette, TDC::OrgPalette*

RestorePen

```
void RestorePen();
```

Restores the original GDI pen object to this DC.

See also: *::SelectObject, TDC::OrgPen*

RestoreTextBrush

```
void RestoreTextBrush();
```



Restores the original GDI text brush object to this DC.

See also: *::SelectObject*

RoundRect

```
inline BOOL RoundRect(int x1, int y1, int x2, int y2, int x3, int y3);
```

```
inline BOOL RoundRect(const TPoint& p1, const TPoint& p2,
                     const TPoint& rad);
```

```
inline BOOL RoundRect(const TPoint& p, const TSize& s, const TPoint& rad);
```

```
inline BOOL RoundRect(const TRect& rect, const TPoint& rad);
```

Draws and fills a rounded rectangle of the given size on this DC with the current pen and brush objects. The current position is neither used nor altered by this call. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *TDC::Rectangle, ::RoundRect*

SaveDC

```
inline virtual int SaveDC() const;
```

Saves the current state of this DC on a context stack. The saved state can be restored later with *RestoreDC()*. Returns a value specifying the saved DC or 0 if the call fails.

See also: *TDC::RestoreDC, ::SaveDC*

ScaleViewportExt

```
inline virtual BOOL ScaleViewportExt(int xNum, int xDenom, int yNum,
                                     int yDenom, TSize* oldExtent = 0);
```

Modifies this DC's viewport extents relative to the current values. The new extents are derived as follows:

$$xNewVE = (xOldVE * xNum) / xDenom$$

$$yNewVE = (yOldVE * yNum) / yDenom$$

The previous extents are saved in *oldExtent*. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *::ScaleViewportExt, TDC::SetViewportExt*

ScaleWindowExt

```
inline virtual BOOL ScaleWindowExt(int xNum, int xDenom, int yNum,
                                   int yDenom, TSize* oldExtent = 0);
```

Modifies this DC's window extents relative to the current values. The new extents are derived as follows:

$$xNewWE = (xOldWE * xNum) / xDenom$$

$$yNewWE = (yOldWE * yNum) / yDenom$$

The previous extents are saved in *oldExtent*. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *TDC::SetWindowExt*, *::ScaleWindowExt*

ScrollDC

```
inline BOOL ScrollDC(int x, int y, const TRect& scroll, const TRect& clip,
                    TRegion& updateRgn, TRect& updateRect);
inline BOOL ScrollDC(const TPoint& delta, const TRect& scroll,
                    const TRect& clip, TRegion& updateRgn,
                    TRect& updateRect);
```

Scrolls a rectangle of bits horizontally by *x* (or *delta.x* in the second version) device-units, and vertically by *y* (or *delta.y*) device-units on this DC. The scrolling and clipping rectangles are specified by *scroll* and *clip*. *ScrollDC* provides data in the *updateRgn* argument telling you the region (not necessarily rectangular) that was uncovered by the scroll. Similarly, *ScrollDC* reports in *updateRect* the rectangle (in client coordinates) that bounds the scrolling update region. This is the largest area that requires repainting.

Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *::ScrollDC*, *::ScrollWindow*, *::InvalidateRegion*

SelectClipPath

```
inline BOOL SelectClipPath(int mode);
```



Selects the current path on this DC as a clip region, combining any existing clip region using the specified *mode* as shown in the following table:

Mode	Meaning
RGN_AND	The new clip region includes the overlapping areas of the current clip region and the current path (intersection).
RGN_COPY	The new clip region is the current path.
RGN_DIFF	The new clip region includes the areas of the current clip region with those of the current path excluded.
RGN_OR	The new clip region includes the combined areas of the current clip region and the current path (union).

RGN_XOR The new clip region includes the combined areas of the current clip region and the current path but without the overlapping areas.

Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *::SelectClipPath*

SelectClipRgn

```
inline int SelectClipRgn(const TRegion& region);
```

Selects the given *region* as the current clipping region for this DC. A copy of the given *region* is used, letting you select the same region for other DC objects. The return value indicates the new clipping region's type as follows:

Region	Meaning
COMPLEXREGION	Clipping Region has overlapping borders.
ERROR	Invalid DC.
NULLREGION	Clipping region is empty.
SIMPLEREGION	Clipping region has no overlapping borders.

See also: *TDC::OffsetClipRgn*, *TDC::GetClipBox*, *::SelectClipRgn*

SelectObject

```
void SelectObject(const TBrush& brush);
void SelectObject(const TPen& pen);
virtual void SelectObject(const TFont& font);
void SelectObject(const TPalette& palette, BOOL forceBackground = FALSE);
```

Selects the given GDI object into this DC. The previously selected object is saved in the appropriate *OrgXXX* data member. For a palette argument, if *forceBackground* is set FALSE (the default), the selected logical palette is a foreground palette when the window has input focus. If *forceBackground* is TRUE, the selected palette is always a background palette whether the window has focus or not.

See also: *::SelectObject*, *::SelectPalette*, *TDC::OrgXXX*, *TMemoryDC::SelectObject*,

SelectStockObject

```
virtual void SelectStockObject(int index);
```

Selects into a DC a predefined stock pen, brush, font, or palette.

See also: *TPrintPreviewDC::SelectStockObject*

SetBkColor

```
inline virtual TColor SetBkColor(TColor color);
```

Sets the current background color of this DC to the given *color* value or the nearest available. Returns 0x80000000 if the call fails.

See also: *TDC::GetBkColor*, *::SetBkColor*

SetBkMode

```
inline int SetBkMode(int mode);
```

Sets the background mode to the given *mode* argument, OPAQUE or TRANSPARENT. Returns the previous background mode.

See also: *TDC::GetBkMode*, *::SetBkMode*

SetBoundsRect

```
inline UINT SetBoundsRect(TRect& bounds, UINT flags);
```

Controls the accumulation of bounding-rectangle information for this DC. Depending on the value of *flags*, the given *bounds* rectangle (possibly NULL) can combine with or replace the existing accumulated rectangle. *flags* can be any appropriate combination of the following values:

Constant	Meaning
DCB_ACCUMULATE	Add <i>bounds</i> (rectangular union) to the current accumulated rectangle.
DCB_DISABLE	Turn off bounds accumulation.
DCB_ENABLE	Turn on bounds accumulation (the default setting for bounds accumulation is disabled).
DCB_RESET	Set the bounding rectangle empty.
DCB_SET	Set the bounding rectangle to <i>bounds</i> .

There are two bounding-rectangle accumulations, one for Windows and one for the application. The Windows-accumulated bounds can be queried by an application but not altered. The application can both query and alter the DC's accumulated bounds.

See also: *TDC::GetBoundsRect*, *::SetBoundsRect*

SetBrushOrg

```
inline BOOL SetBrushOrg(const TPoint& origin, TPoint* oldOrg = 0);
```

Sets the origin of the currently selected brush of this DC with the given *origin* value. The previous origin is passed to *oldOrg*. Returns TRUE if successful; otherwise returns FALSE.

See also: *TDC::GetBrushOrg*

SetDIBits

```
inline BOOL SetDIBits(TBitmap& bitmap, WORD startScan, WORD numScans,
                    const void HUGE* bits, const BITMAPINFO far& Info,
                    WORD usage);
inline BOOL SetDIBits(TBitmap& Bitmap, const TDib& dib);
```

The first version sets the pixels in *bitmap* (the given destination bitmap on this DC), from the source DIB (device-independent bitmap) color data found in the byte array *bits* and the BITMAPINFO structure, *Info*. *numScan* scanlines are taken from the DIB, starting at scanline *startScan*. The *usage*

argument specifies how the *bmiColors* member of BITMAPINFO is interpreted, as explained in *TDC::GetDIBits()*.

In the second version of *SetDIBits*, the pixels are set in *bitmap* from the given source *TDib* argument.

SetDIBits returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: *TDC::GetDIBits*, *TDC::SetDIBitsToDevice*, *::SetDIBits*

SetDIBitsToDevice

```
inline BOOL SetDIBitsToDevice(const TRect& dst, const TPoint& src,
                             WORD startScan, WORD numScans,
                             const void HUGE* bits,
                             const BITMAPINFO far& bitsInfo, WORD usage);
inline BOOL SetDIBitsToDevice(const TRect& dst, const TPoint& src,
                             const TDib& dib);
```

The first version sets the pixels in *dst* (the given destination rectangle on this DC) from the source DIB (device-independent bitmap) color data found in the byte array *bits* and the BITMAPINFO structure, *bitsInfo*. The DIB origin is specified by the point *src*. *numScan* scanlines are taken from the DIB, starting at scanline *startScan*. The *usage* argument determines how the *bmiColors* member of BITMAPINFO is interpreted, as explained in *TDC::GetDIBits()*.

In the second version of *SetDIBitsToDevice*, the pixels are set in *dst* from *dib*, the given source *TDib* argument.

SetDIBits returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: *TDC::GetDIBits*, *::SetDIBitsToDevice*, *TDib*

SetMapMode

```
inline virtual int SetMapMode(int mode);
```

Sets the current window mapping mode of this DC to *mode*. Returns the previous mapping mode value. The mapping mode defines how logical coordinates are mapped to device coordinates. It also controls the orientation of the device's x- and y-axes. See *TDC::GetMapMode* for a complete list of mapping modes.

See also: *TDC::GetMapMode*, *::SetMapMode*

SetMapperFlags

```
inline DWORD SetMapperFlags(DWORD flag);
```

Alters the algorithm used by the font mapper when mapping logical fonts to physical fonts on this DC. If successful, the function sets the current font-mapping flag to *flag* and returns the previous mapping flag; otherwise GDI_ERROR is returned. The mapping flag determines whether the font mapper will attempt to match a font's aspect ratio to this DC's aspect ratio. If bit 0 of *flag* is set to 1, the mapper selects only matching fonts. If no

matching fonts exist, a new aspect ratio is chosen and a font is retrieved to match this ratio.

See also: `::SetMapperFlags`

SetMiterLimit



```
inline BOOL SetMiterLimit(float newLimit, float* oldLimit = 0);
```

Sets the limit of miter joins to *newLimit* and puts the previous value in *oldLimit*. Returns TRUE if successful; otherwise returns FALSE.

See also: `::SetMiterLimit`

SetPixel

```
inline TColor SetPixel(int x, int y, TColor color);
inline TColor SetPixel(const TPoint& p, TColor color);
```

Sets the color of the pixel at the given location to the given color and returns the pixel's previous color.

See also: `TDC::GetPixel`, `::SetPixel`

SetPolyFillMode

```
inline int SetPolyFillMode(int mode);
```

Sets the polygon-filling mode for this DC to the given mode value, either ALTERNATE or WINDING. Returns the previous fill mode.

See also: `TDC::GetPolyFillMode`, `::SetPolyFillMode`, `TDC::Polygon`

SetROP2

```
inline int SetROP2(int mode);
```

Sets the current foreground mix mode mode of this DC to the given *mode* value and returns the previous mode. The *mode* argument determines how the brush, pen, and existing screen image combine when filling and drawing. *mode* can be one of the following values:

Value	Meaning
R2_BLACK	Pixel is always binary 0.
R2_COPYPEN	Pixel is the pen color.
R2_MASKNOTPEN	Pixel is a combination of the colors common to both the display and the inverse of the pen.
R2_MASKPEN	Pixel is a combination of the colors common to both the pen and the display.
R2_MASKPENNOT	Pixel is a combination of the colors common to both the pen and the inverse of the display.
R2_MERGE PEN	Pixel is a combination of the pen color and the display color.
R2_MERGE NOTPEN	Pixel is a combination of the display color and the inverse of the pen color.
R2_MERGE PENNOT	Pixel is a combination of the pen color and the inverse of the display color.
R2_NOP	Pixel remains unchanged.
R2_NOT	Pixel is the inverse of the display color.
R2_NOTCOPYPEN	Pixel is the inverse of the pen color.

Value	Meaning
R2_NOTMASKPEN	Pixel is the inverse of the R2_MASKPEN color.
R2_NOTMERGEPEN	Pixel is the inverse of the R2_MERGEPEN color.
R2_NOTXORPEN	Pixel is the inverse of the R2_XORPEN color.
R2_WHITE	Pixel is always binary 1.
R2_XORPEN	Pixel is a combination of the colors in the pen and in the display, but not in both.

See also: *TDC::GetROP2*, *::SetROP2*, *TDC::GetDeviceCaps*

SetStretchBltMode inline int SetStretchBltMode(int mode);

Sets the stretching mode of this DC to the given *mode* value and returns the previous mode. The *mode* argument defines which scan lines and/or columns are eliminated by *TDC::StretchBlt()*: BLACKONWHITE, COLORONCOLOR, or WHITEONBLACK.

See also: *TDC::GetStretchBltMode*, *::SetStretchBltMode*, *TDC::StretchBlt*

SetSystemPaletteUse inline int SetSystemPaletteUse(int usage);

Changes the usage of this DC's system palette. The *usage* argument can be SYSPAL_NOSTATIC or SYSPAL_STATIC. Returns the previous usage value.

See also: *TDC::GetSystemPaletteUse*, *::SetSystemPaletteUse*

SetTextAlign inline UINT SetTextAlign(UINT flags);

Sets the text-alignment flags for this DC. If successful, *SetTextAlign* returns the previous text-alignment flags; otherwise, it returns GDI_ERROR. The flag values are as listed for the *TDC::GetTextAlign* function. The text-alignment flags determine how *TDC::TextOut()* and *TDC::ExtTextOut* align text strings in relation to the first character's screen position.

See also: *TDC::GetTextAlign*, *::SetTextAlign*, *TDC::TextOut*, *TDC::ExtTextOut*

GetTextCharacterExtra inline int GetTextCharacterExtra() const;

If successful, returns the current intercharacter spacing, in logical units, for this DC; otherwise returns INVALID_WIDTH.

See also: *TDC::SetTextCharacterExtra*, *::GetTextCharacterExtra*

SetTextCharacterExtra inline int SetTextCharacterExtra(int extra);

If successful, sets the current intercharacter spacing to *extra*, in logical units, for this DC, and returns the previous intercharacter spacing. Otherwise,

returns 0. If the current mapping mode is *not* MM_TEXT, the *extra* value is transformed and rounded to the nearest pixel.

See also: *TDC::GetTextCharacterExtra*, *::SetTextCharacterExtra*

SetTextColor

```
inline virtual TColor SetTextColor(TColor color);
```

Sets the current text color of this DC to the given *color* value. The text color determines the color displayed by *TDC::TextOut()* and *TDC::ExtTextOut()*.

See also: *TDC::GetTextColor*, *::SetTextColor*

SetTextJustification

```
inline BOOL SetTextJustification(int breakExtra, int breakCount);
```

Sets *breakExtra* logical units as the total extra space to be added to *breakCount* break characters when text strings are displayed using *TDC::TextOut()* and *TDC::ExtTextOut()*. The extra space is distributed evenly between the break characters. The break character is usually the ASCII 32 space, but some fonts define other characters.

TDC::GetTextMetrics() can be used to retrieve the value of the break character. *TDC::GetTextExtentPoint()* must be called to obtain text width before justification. From this, the *breakExtra* value can be computed for the *SetTextJustification* call.

If the current mapping mode is not MM_TEXT, the extra value is transformed and rounded to the nearest pixel.

SetTextJustification returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: *::SetTextJustification*, *TDC::GetTextExtentPoint*

SetViewportExt

```
inline virtual BOOL SetViewportExt(const TSize& extent, TSize* oldExtent = 0);
```

Sets this DC's viewport x- and y-extents to the given *extent* values. The previous extents are saved in *oldExtent*. Returns TRUE if the call is successful; otherwise, returns FALSE. The *extent* value determines the amount of stretching or compression needed in the logical coordinate system to fit the device coordinate system. *extent* also determines the relative orientation of the two coordinate systems.

See also: *TDC::GetViewportExt*, *::SetViewportExt*

SetViewportOrg

```
inline virtual BOOL SetViewportOrg(const TPoint& origin, TPoint* oldOrg = 0);
```

Sets this DC's viewport origin to the given *origin* value, and saves the previous origin in *oldOrg*. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *TDC::GetViewportOrg*, *TDC::OffsetViewportOrg*, *::SetViewportOrg*

SetWindowExt

```
inline virtual BOOL SetWindowExt(const TSize& extent, TSize* oldExtent
                                = 0);
```

Sets this DC's window x- and y-extents to the given *extent* values. The previous extents are saved in *oldExtent*. Returns TRUE if the call is successful; otherwise, returns FALSE. The *extent* value determines the amount of stretching or compression needed in the logical coordinate system to fit the device coordinate system. *extent* also determines the relative orientation of the two coordinate systems.

See also: *TDC::GetWindowExt*, *::SetWindowExt*, *TDC::ScaleWindowExt*

SetWindowOrg

```
inline BOOL SetWindowOrg(const TPoint& origin, TPoint* oldOrg = 0);
```

Sets the origin of the window associated with this DC to the given *origin* value, and saves the previous origin in *oldOrg*. Returns TRUE if the call is successful; otherwise, returns FALSE.

See also: *TDC::GetWindowOrg*, *TDC::OffsetWindowOrg*, *::SetWindowOrg*

SetWorldTransform

```
inline BOOL SetWorldTransform(XFORM far& xform);
```



Sets a two-dimensional linear transformation, given by the *xform* structure, between world space and page space for this DC. Returns TRUE if the call is successful; otherwise, returns FALSE.

See also: *TDC::ModifyWorldTransform*, *::SetWorldTransform*, *XFORM*

StretchBlt

```
inline BOOL StretchBlt(int dstX, int dstY, int dstW, int dstH,
                      const TDC& srcDC, int srcX, int srcY, int srcW,
                      int srcH, DWORD rop);
```

```
inline BOOL StretchBlt(const TRect& dst, const TDC& srcDC,
                      const TRect& src, DWORD rop);
```

Copies a bitmap from the source DC to a destination rectangle on this DC specified either by upper left-corner coordinates (*dstX*, *dstY*), width *dstW*, and height *dstH*, or (in the second version) by a *TRect* object, *dst*. The source bitmap is similarly specified with (*srcX*, *srcY*), *srcW*, and *srcH*, or by a *TRect* object, *src*. *StretchBlt* stretches or compresses the source according to the stretching mode currently set in this DC (the destination DC). The raster-operation code, *rop*, specifies how the colors are combined in output operations that involve a brush, a source bitmap, and a destination bitmap. The *rop* codes are described in the entry for *TDC::MaskBlt*.

See also: *::StretchBlt*, *TDC::MaskBlt*, *TDC::SetStretchBltMode*

StretchDIBits

```
inline BOOL StretchDIBits(const TRect& dst, const TRect& src,
                        const void HUGE* bits,
                        const BITMAPINFO far& bitsInfo, WORD usage,
                        DWORD rop);
inline BOOL StretchDIBits(const TRect& dst, const TRect& src,
                        const TDib& dib, DWORD rop);
```

Copies the color data from *src*, the source rectangle of pixels in the given DIB (device-independent bitmap) on this DC, to *dst*, the destination rectangle. The DIB bits and color data are specified in either the byte array *bits* and the BITMAPINFO structure *bitsInfo* or in the TDib object, *dib*. The rows and columns of color data are stretched or compressed to match the size of the destination rectangle. The *usage* argument specifies how the *bmiColors* member of BITMAPINFO is interpreted, as explained in *TDC::GetDIBits()*. The raster operation code, *rop*, specifies how the source pixels, the current brush for this DC, and the destination pixels are combined to produce the new image. See *TDC::MaskBlt* for a detailed list of *rop* codes.

See also: *::StretchDIBits*, *TDC::MaskBlt*, *TDib*

StrokeAndFillPath

```
inline BOOL StrokeAndFillPath();
```



Closes any open figures in the current path of this DC, strokes the outline of the path using the current pen, and fills its interior using the current brush and polygon fill mode. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *TDC::StrokePath*, *::StrokeAndFillPath*, *TDC::BeginPath*, *TDC::FillPath*, *TDC::EndPath*, *TDC::SetPolyFillMode*,

StrokePath

```
inline BOOL StrokePath();
```



Renders the current, closed path on this DC, using the DC's current pen.

See also: *::StrokePath*, *TDC::StrokeAndFillPath*, *TDC::BeginPath*

TabbedTextOut

```
inline virtual BOOL TabbedTextOut(const TPoint& p, const char far* string,
                                int count, int numPositions,
                                const int* positions, int tabOrigin);
inline BOOL TabbedTextOut(const TPoint& p, const char far* string,
                        int count, int numPositions,
                        const int* positions, int tabOrigin,
                        TSize& size);
```

Draws up to *count* characters of the given null-terminated *string* in the current font on this DC. If *count* is -1, the whole string is written.

Tabs are expanded according to the given arguments. The *positions* array specifies *numPositions* tab stops given in device units. The tab stops must have strictly increasing values in the array. If *numPositions* and *positions* are both 0, tabs are expanded to eight times the average character width. If *numPositions* is 1, all tab stops are taken to be *positions[0]* apart. *tabOrigin* specifies the x-coordinate in logical units from which tab expansion will start.

The *p* argument specifies the logical coordinates of the reference point that is used to align the first character depending on the current text-alignment mode. This mode can be inspected with *TDC::GetTextAlign* and changed with *TDC::SetTextAlign*. By default, the current position is neither used nor altered by *TabbedTextOut*. However, if the align mode is set to *TA_UPDATECP*, *TabbedTextOut* ignores the reference point argument(s) and uses/updates the current position as the reference point.

The *size* argument in the second version of *TabbedTextOut* reports the dimensions (*size.x* = height and *size.y* = width) of the string in logical units.

TabbedTextOut returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: *TDC::TextOut*, *::TabbedTextOut*

TextOut

```
inline virtual BOOL TextOut(int x, int y, const char far* string,
                           int count = -1);
inline BOOL TextOut(const TPoint& p, const char far* string, int count
                   = -1);
```

Draws up to *count* characters of the given null-terminated *string* in the current font on this DC. If *count* is -1 (the default), the entire string is written.

The (*x*, *y*) or *p* arguments specify the logical coordinates of the reference point that is used to align the first character, depending on the current text-alignment mode. This mode can be inspected with *TDC::GetTextAlign* and changed with *TDC::SetTextAlign*. By default, the current position is neither used nor altered by *TextOut*. However, the align mode can be set to *TA_UPDATECP*, which makes Windows use and update the current position. In this mode, *TextOut* ignores the reference point argument(s).

TextOut returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: *TDC::ExtTextOut*, *::TextOut*, *TDC::GetTextAlign*

TextRect

```
inline BOOL TextRect(int x1, int y1, int x2, int y2);
inline BOOL TextRect(const TRect& rect);
inline BOOL TextRect(int x1, int y1, int x2, int y2, TColor color);
inline BOOL TextRect(const TRect rect, TColor color);
```

Fills the given rectangle by calling `::ExtTextOut` with an `ETO_OPAQUE` rectangle type argument and a NULL string. If no color argument is supplied, the current background color is used. If a color argument is supplied, that color is set to the current background color which is then used for filling. `TextRect` returns TRUE if the call is successful; otherwise, returns FALSE.

See also: `::ExtTextOut`, `TDC::SetBkColor`, `ETO_OPAQUE`,

UpdateColors

```
inline void UpdateColors();
```

Updates the client area of this DC by matching the current colors in the client area to the system palette on a pixel-by-pixel basis.

See also: `::UpdateColors`

WidenPath

```
inline BOOL WidenPath();
```



Redefines the current, closed path on this DC as the area that would be painted if the path were stroked with this DC's current pen. The current pen must have been created under the following conditions:

If `::CreatePen` or the `TPen::TPen(int Style, int Width, TColor Color)` or `TPen::TPen(const LOGPEN* LogPen)` constructors were used, the width of the pen in device units must be greater than 1.

If `::ExtCreatePen` or the `TPen::TPen(DWORD PenStyle, DWORD Width, const TBrush& Brush, DWORD StyleCount, LPDWORD pStyle)` or `TPen::TPen(DWORD PenStyle, DWORD Width, const LOGBRUSH& logBrush, DWORD StyleCount, LPDWORD pStyle)` constructors were used, the pen must be a geometric pen.

Any Bezier curves in the path are converted to sequences of linear segments approximating the widened curves, so no Bezier curves remain in the path after a `WidenPath` call.

`WidenPath` returns TRUE if the call is successful; otherwise, it returns FALSE.

See also: `::WidenPath`, `TDC::FlattenPath`, `TDC::BeginPath`

Protected data members

Handle

HDC Handle;

The Windows handle of this DC.

See also: `TDC` constructors

OrgBrush

HBRUSH OrgBrush;

Handle to the original GDI brush object for this DC. Holds the previous brush object whenever a new brush is selected with *SelectObject(brush)*.

See also: *TDC::SelectObject, ::SelectObject*

OrgFont

HFONT OrgFont;

Handle to the original GDI font object for this DC. Holds the previous font object whenever a new font is selected with *SelectObject(font)*.

See also: *TDC::SelectObject, ::SelectObject*

OrgPalette

HPALETTE OrgPalette;

Handle to the original GDI palette object for this DC. Holds the previous palette object whenever a new palette is selected with *SelectObject(palette)*.

See also: *TDC::SelectObject, ::SelectPalette*

OrgPen

HPEN OrgPen;

Handle to the original GDI pen object for this DC. Holds the previous pen object whenever a new pen is selected with *SelectObject(pen)*.

See also: *TDC::SelectObject, ::SelectObject*

OrgTextBrush

HBRUSH OrgTextBrush;



The handle to the original GDI text brush object for this DC. Stores the previous text brush handle whenever a new brush is selected with *SelectObject(text_brush)*.

See also: *TDC::SelectObject, ::SelectObject*

ShouldDelete

BOOL ShouldDelete;

Set to TRUE if Handle for this object should be deleted by the destructor; otherwise set to FALSE.

See also: *enum TDC::TAutoDelete, ~TDC*

Protected constructors

Constructor

TDC();

For use by derived classes only. Calls *Init* to clear the *OrgXXX* data members and sets *ShouldDelete* to TRUE.

See also: *TDC::Init*

Protected member functions

GetAttributeHDC

```
virtual HDC GetAttributeHDC() const;
```

Returns the attributes of the DC object.

See also: *TPrintPreview::GetAttributeHDC*

GetHDC

```
inline HDC GetHDC() const;
```

Returns a handle to the DC.

Init

```
void Init();
```

Sets *OrgBrush*, *OrgPen*, *OrgFont*, *OrgBitmap*, and *OrgPalette* to 0, and sets *ShouldDelete* to TRUE. This function is for internal use by the TDC constructors.

See also: *TDC constructors*, *SelectObject*

TDecoratedFrame class

deccframe.h

TDecoratedFrame automatically positions its client window (you must supply a client window) so that it is the same size as the client rectangle. You can add additional decorations like toolbars and status lines to a window. *TDecoratedFrame* is a streamable class. For more information about decorated frame windows, see Chapter 6 in the *ObjectWindows Programmer's Guide*.

Public constructors

Constructor

```
TDecoratedFrame(TWindow* parent, const char far *title, TWindow*
    clientWnd, BOOL trackMenuSelection = FALSE, TModule*
    module = 0);
```

Constructs a *TDecoratedFrame* object with the specified parent window (*parent*), window caption (*title*), and module ID. Sets *Attr.Title* to the new title. Passes a pointer to the client window if one is specified. By default set to FALSE, *trackMenuSelection* controls whether hint text appears at the bottom of the window when a menu item is highlighted.

Public member functions

Insert

```
void Insert (TWindow& decoration, TLocation = Top);
```

After you specify where the decoration should be placed, *Insert* adds it just above, below, left, or right of the client window. This process is especially important when there are multiple decorations. *Insert* looks at the decoration's *Attr.Style* member and checks the `WS_VISIBLE` flag to tell whether the decoration should initially be visible or hidden. To position the decoration, *Insert* uses *TLocation enum*, which describes *Top*, *Left*, *Bottom*, and *Right* positions where the decoration can be placed.

PreProcessMsg

```
BOOL PreProcessMsg (MSG& msg);
```

Overrides the virtual function defined in *TFrameWindow* to give decorations an opportunity to perform mnemonic access preprocessing.

See also: *TFrameWindow::PreProcessMsg*, *TWindow::PreProcessMsg*

Protected data members

MenuItemId

```
UINT MenuItemID;
```

Specifies the menu item ID.

TrackMenuSelection

```
BOOL TrackMenuSelection;
```

Specifies whether you want menu selection and help status information visible.

Protected member functions

EvCommand

```
LRESULT EvCommand(UINT Id, HWND hWndCtl, UINT notifyCode);
```

Automates hiding and showing of decorations.

EvCommandEnable

```
void EvCommandEnable(TCommandEnabler& ce);
```

Handles checking and unchecking of menu items that are associated with decorations.

See also: *TMenuItemEnabler::TCommandEnabler*

EvEnterIdle

```
void EvEnterIdle(UINT source, HWND hWndDlg);
```

Responds to a Windows API `WM_ENTERIDLE` message that tells an application's main window that a dialog box or a menu is entering an idle state. *EvEnterIdle* also handles updating the status bar with the appropriate help message.

See also: `::WM_ENTERIDLE`

EvMenuSelect

```
void EvMenuSelect(UINT MenuItemId, UINT flags, HMENU hMenu);
```

Responds to user menu selection. If *MenuItemId* is blank, displays an empty help message; otherwise, it displays a help message with the specified string ID. See *EvEnterIdle* for a description of how the help message is loaded.

EvSize

```
inline void EvSize(UINT sizeType, TSize& size);
```

Passes a WM_SIZE message to *TLayoutWindow*.

See also: *TWindow::EvSize*

SetupWindow

```
void SetupWindow();
```

Calls *Layout* to size and position the decoration.

See also: *TFrameWindow::SetUpWindow*, *TWindow::SetUpWindow*

Response table entries

Response table entry	Member function
EV_WM_ENTERIDLE	EvEnterIdle
EV_WM_MENUSELECT	EvMenuSelect
EV_WM_SIZE	EvSize

TDecoratedMDIFrame class

decmdifr.h

TDecoratedMDIFrame is an MDI frame that supports decorated child windows. *TDecoratedMDIFrame* is a streamable class.

Public constructors**Constructor**

```
TDecoratedMDIFrame(const char far *title, TResId menuResId,
    TMDIClient &clientWnd = *new TMDIClient,
    BOOL trackMenuSelection = FALSE, TModule* module = 0);
```

Constructs a decorated MDI frame of the specified client window with the indicated menu resource ID. By default, menu hint text is not displayed.

Protected member function**DefWindowProc**

```
LRESULT DefWindowProc(UINT message, WPARAM wParam, LPARAM lParam);
```

Overrides *TWindow::DefWindowProc* and calls the Windows API function *::DefFrameProc* that provides default processing for any incoming message the MDI child window does not process.

See also: *::DefFrameProc*, *::DefWindowProc*, *TMDIFrame::DefWindowProc*

Response table entries

The *TDecoratedMDIFrame* response table has no entries.

TDesktopDC class

dc.h

A DC class derived from *TWindowDC*, *TDesktopDC* provides access to the desktop window's client area, which is the screen behind all other windows.

Public constructors

Constructor

TDesktopDC();

Default constructor for *TDesktopDC* objects.

TDialog class

dialog.h

See Chapter 4 in the *ObjectWindows Programmer's Guide* for a description of interface objects. See Chapter 8 in the *ObjectWindows Programmer's Guide* for specific instructions about creating dialog boxes.

TDialog objects represent both modal and modeless dialog box interface elements. (A modal dialog box disables operations in its parent window while it is open.) A *TDialog* object has a corresponding resource definition that describes the placement and appearance of its controls. The identifier of this resource definition is supplied to the constructor of the *TDialog* object. A *TDialog* object is associated with a modal or modeless interface element by calling its *Execute* or *Create* member function, respectively. *TDialog* is a streamable class.

ObjectWindows provides three-dimensional (3-D) support for dialog boxes. If your application expects to use Microsoft's CTL3D DLL, you need to register your application by calling *TApplication::EnableCtl3d*. ObjectWindows will then automatically forward the WM_CTLCOLOR message to the CTL3D DLL.

ObjectWindows also provides BWCC support for dialog boxes. Unless a custom template is specified, *TDialog* uses the BWCC templates. (By

default, *TApplication's* member function *EnableBWCC* enables BWCC support.)

Public data members

Attr

TDialogAttr Attr;

Attr holds the dialog creation attributes of the dialog box. *TDialogAttr* is defined on page 156.

IsModal

BOOL IsModal;

IsModal is TRUE if the dialog box is modal and FALSE if it is modeless.

Public constructors and destructor

Constructor

TDialog(TWindow* parent, TResId resId, TModule* module = 0);

Invokes a *TWindow* constructor, passing *parent* and *module*, and calls *DisableAutoCreate* so that the *TDialog* will not be automatically created and displayed along with its parent. *TDialog* then initializes *Title* to -1 and sets *Attr.Name* using the dialog's integer or string resource identifier, which must correspond to a dialog resource definition in the resource file. Finally, it initializes *Attr.Param* to 0 and sets *IsModal* to FALSE.

See also: *TApplication::EnableBWCC*, *TWindow::TWindow*, *TWindow::DisableAutoCreate*

Destructor

~TDialog();

If *Attr.Name* is a string and not an integer resource identifier, this destructor frees memory allocated to hold the name of the *TDialog* (*Attr.Name*).

See also *TWindow::~TWindow*

Public member functions

CloseWindow

void CloseWindow(int retValue = IDCANCEL);

Conditionally shuts down the dialog box. If the dialog box is modeless, it calls *TWindow::CloseWindow*. If the dialog box is modal, it calls *CanClose*. If *CanClose* returns TRUE, *CloseWindow* calls *TransferData* to transfer dialog box data, passing it *retValue*. The default value of *retValue* is IDCANCEL.

See also: *TWindow::CloseWindow*

CmCancel

void CmCancel();

Automatic response to a click on the Cancel button of the dialog box. Calls *CloseWindow*, passing IDCANCEL.

See also *TDialog::CloseWindow*

CmOk

```
void CmOk();
```

Responds to a click on the dialog box's OK button (with the identifier IDOK). Calls *CloseWindow*, passing IDOK.

See also: *TDialog::CloseWindow*

Create

```
virtual BOOL Create();
```

Creates a modeless dialog box interface element associated with the *TDialog* object. Calls *DisableAutoCreate* to prevent automatic creation of all child windows. Calls *EnableKBHandler* to enable keyboard handling. Registers all the dialog's child windows for custom control support. Calls the Windows function *CreateDialogParam* to create the dialog box.

Create returns TRUE if successful. If unsuccessful, *Create* throws a *TXInvalidWindow* exception.

See also: *TDialog::Execute*, *TModule::MakeWindow*, *TWindow::DisableAutoCreate*

Destroy

```
virtual void Destroy(int retValue = IDCANCEL);
```

Destroys the interface element associated with the *TDialog* object. If the element is a modeless dialog box, *Destroy* calls *TWindow::Destroy*. If the element is a modal dialog box, *Destroy* calls *EnableAutoCreate* on all child windows. Then *Destroy* calls the Windows function *EndDialog*, passing *retValue* as the value returned to indicate the result of the dialog's execution. The default *retValue* is IDCANCEL.

See also: *TWindow::Destroy*, *TWindow::EnableAutoCreate*

DialogFunction

```
virtual BOOL DialogFunction(UINT message, WPARAM, LPARAM);
```

To process messages within the dialog function, your application must override this function. *DialogFunction* returns TRUE if the message is handled and FALSE if the message is not handled.

DoCreate

```
virtual HWND DoCreate();
```

Create calls *DoCreate* to perform the actual creation of a dialog box. *DoCreate* calls the Windows API function *CreateDialogParam* to perform the actual creation of the dialog box.

DoExecute

```
virtual int DoExecute();
```

Execute calls *DoExecute*, which calls the Windows API function *DialogBoxParam* to perform the actual execution of a dialog box.

EvClose

```
void EvClose();
```

Responds to an incoming *EvClose* message by shutting down the window.

EvInitDialog

```
virtual BOOL EvInitDialog(HWND hWndFocus);
```

EvInitDialog is automatically called just before the dialog box is displayed. It calls *SetupWindow* to perform any setup required for the dialog box or its controls.

See also: *TWindow::SetupWindow*

EvPaint

```
void EvPaint();
```

If the control has a predefined Windows class, *EvPaint* calls *DefWndProc* for Windows-supplied painting. Otherwise, it calls *TWindow::EvPaint*.

See also: *TWindow::EvPaint*

EvSetFont

```
virtual void EvSetFont(HFONT);
```

Responds to a request to change a dialog's font.

Execute

```
virtual int Execute();
```

Execute creates and executes a modal dialog box interface element associated with the *TDialog* object. If the element is successfully associated, *Execute* does not return until the *TDialog* is closed.

Execute then calls *DisableAutoCreate* to prevent all child windows from being created automatically. After it calls *EnableKBHandler* to enable keyboard handling, *DialogBoxParam* registers all the dialog's child windows for custom control support. Finally, *Execute* calls *DoExecute*, which calls the Windows API function *DialogBoxParam* to execute the dialog box. If errors occur, a *TXInvalidWindow* exception is thrown.

See also: *TModule::ExecDialog*, *TWindow::DisableAutoCreate*

GetDefaultId

```
UINT GetDefaultId() const;
```

Gets the default resource ID.

GetItemHandle

```
HWND GetItemHandle(int childId);
```

Returns the dialog box control's window handle identified by the supplied ID. Because *GetItemHandle* is now obsolete, new applications should use *TWindow::GetDlgItem*.

PreProcessMsg

```
BOOL PreProcessMsg(MSG &);
```

Performs preprocessing of window messages. If the child window has requested keyboard navigation, *PreProcessMsg* handles any accelerator key messages and then calls the Windows API function *::IsDialogMessage* to process any other keyboard messages.

See also: *TWindow::PreProcessMsg*

SendDlgItemMsg `DWORD SendDlgItemMsg(int ChildId, WORD Msg, WORD WParam, DWORD LParam);`

Sends a Windows control message, identified by *Msg*, to the dialog box's control identified by its supplied ID, *ChildID*. *WParam* and *LParam* become parameters in the control message. *SendDlgItemMsg* returns the value returned by the control, or 0 if the control ID is invalid.

SetCaption `void SetCaption(const char far* title);`

If *Title* is not `-1`, calls *TWindow::SetCaption*.

See also: *SetupWindow*, *TWindow::SetCaption*

SetDefaultId `void SetDefaultId(UINT Id);`

Sets the default resource ID.

Protected member functions

EvCtlColor `HBRUSH EvCtlColor(HDC, HWND hWndChild, UINT ctlType);`

Passes the handle to the display context for the child window, the handle to the child window, and the default system colors to the parent window. The parent window then uses the display-context handle given in *HDC* to set the text and background colors of the child window.

If three-dimensional (3-D) support is enabled, *EvCtlColor* handles the *EV_WM_CTLCOLOR* message by allowing the *CTL3D* DLL to process the *WM_CTLCOLOR* message in order to set the background color and provide a background brush for the window.

See also: *TApplication::EnableCtl3d*

GetClassName `char far* GetClassName();`

GetClassName overrides the virtual function defined in *TWindow* and returns the name of the dialog box's default Windows class, which must be used for a modal dialog box. For a modeless dialog box, *GetClassName* returns the name of the default *TWindow*. If *BWCC* is enabled, *GetClassName* returns *BORDLGCLASS*.

See also: *TWindow::GetClassName*

GetWindowClass void GetWindowClass(WNDCLASS& wndClass);

Overrides the virtual function defined in *TWindow*. Fills *WndClass* with a *TDialog* registration attributes obtained from an existing *TDialog* window or BWCC if enabled. By default, a *TDialog* object has a standard application icon and arrow cursor. To change these or other registration attributes, redefine this member function and *GetClassName* in your *TDialog* derived class. Be sure to call *GetWindowClass* in your *GetWindowClass* member function prior to modifying the defaults that *TDialog* sets.

See also: *TWindow::GetWindowClass*

SetupWindow void SetupWindow();

Overrides the virtual function defined in *TWindow*. Sets up the dialog box by calling *SetCaption* (sets *Title*) and *TWindow::SetupWindow*.

If three-dimensional (3-D) support is enabled, *SetupWindow* calls the CTL3D DLL to create the dialog box.

See also: *TCommonDialog::SetupWindow*, *TDialog::SetCaption*, *TWindow::SetupWindow*

Response table entries

Response table entry	Member function
EV_COMMAND (IDCANCEL, CmCancel)	CmCancel
EV_COMMAND (IDOK, CmOk)	CmOk
EV_CTLCOLOR	EvCtlColor
EV_WM_CLOSE	EvClose
EV_WM_PAINT	EvPaint

TDialogAttr struct

dialog.h

A *TDialogAttr* is used to hold a *TDialog*'s creation attributes.

Public data members

Name char far* Name;

Name holds the identifier of the dialog resource.

Param DWORD Param;

Param holds a parameter that is supplied to the dialog box when it is created. After Windows accepts *Param*, it is then available in the message response functions associated with WM_INITDIALOG.

See also: *TDialog::Attr*

TDib class

gdiobjec.h

The class *TDib*, derived from *TGdiObject*, represents GDI Device Independent Bitmap (DIB) objects. *TDibDCs* encapsulate the creation of DCs using DIB.DRV (a GDI driver provided with Windows MME and 3.1). DIBs have no Windows handle; they are just structures containing format and palette information and a collection of bits or pixels. *TDib* provides a convenient way to work with DIBs like any other GDI object. The memory for the DIB is in one *GlobalAlloc*'d unit so it can be passed to the Clipboard, OLE, and so on.

The *TDib* destructor overloads the base destructor because DIBs are not real GDI objects.

Protected data members

Bits

void HUGE* Bits;

Bits points into the block of memory pointed to by *Info*.

See also: *TDib::GetBits*

H

int H;

The height of this DIB in pixels.

See also: *TDib::Height*, *TDib::Size*, *TDib::NumScans*

Info

BITMAPINFO far* Info;

Locked global allocated block.

See also: *TDib::GetInfo*

IsCore

BOOL IsCore;

Set TRUE if this DIB is an old-style PM DIB using core headers; otherwise, set FALSE.

See also: *TDib::isPM*

Mode

WORD Mode;

If *Mode* is *DIB_RGB_Colors*, the color table contains 4-byte RGB entries. If *Mode* is *DIB_PAL_COLORS*, the color table contains 2-byte indexes into some other palette (such as the system palette). Because either of these two cases might exist, two versions of certain functions (such as *GetColors()* and *GetIndices()*) are required.

See also: *TDib::GetColors*, *TDib::GetIndices*, *TDib::Usage*

NumClrs

long NumClrs;

The number of colors associated with this DIB.

See also: *TDib::NumColors*

W

int W;

The width of this DIB in pixels.

See also: *TDib::Width*, *TDib::Size*

Public constructors and destructor

Constructor

TDib(HGLOBAL handle, TAutoDelete autoDelete = NoAutoDelete);

Creates a *TDib* object and sets the *Handle* data member to the given borrowed *handle*. The *ShouldDelete* data member defaults to *FALSE*, ensuring that the borrowed handle will not be deleted when the C++ object is destroyed.

See also: *TGdiObject::Handle*, *TGdiObject::ShouldDelete*, *TDib::InfoFromHandle*

Constructor

TDib(const TClipboard& clipboard);

Constructs a *TDib* object with a handle borrowed from the given *Clipboard*.

See also: *TDib::InfoFromHandle*, *::GetClipboardData*

Constructor

TDib(const TDib& dib);

This public copy constructor creates a complete copy of the given *dib* object, as in:

```
TDib myDib = yourDib;
```

Constructor

TDib(int width, int height, int nColors, WORD mode = DIB_RGB_COLORS);

Creates a DIB object with the given width, height, number of colors, and mode values.

See also: *DIB_RGB_COLORS*

- Constructor** TDib(HINSTANCE instance, TResID resID);
Creates a DIB object from the resource with the given ID.
See also: *TDib::LoadResource*
- Constructor** TDib(const char* name);
Creates a DIB object from the given resource file.
See also: *TDib::LoadFile*
- Constructor** TDib(const TBitmap& bitmap, const TPalette* pal = 0);
Creates a DIB object from the given resource bitmap and palette. If *pal* is 0 (the default), the default palette is used.
See also: *TScreenDC*
- Destructor** ~TDib();
Overrides the base destructor.
See also: *~TGdiObject*

Public member functions

- ChangeModeToPal** BOOL ChangeModeToPal(const TPalette& pal);
Converts the existing color table in place to use palette relative values. The palette that is passed is used as a reference.
See also: *TDib::ChangeModeToRGB, TPalette::GetPaletteEntry*
- ChangeModeToRGB** BOOL ChangeModeToRGB(const TPalette& pal);
Converts the existing color table in place to use absolute RGB values. The palette that is passed is used as a reference. Returns TRUE if the call is successful; otherwise returns FALSE.
See also: *TDib::ChangeModetoPal, TPalette::GetPaletteEntry*
- FindColor** int FindColor(TColor color);
Returns the palette entry for the given color.
See also: *TDib::GetColor, TColor, TDib::SetColor, TDib::MapColor*
- FindIndex** int FindIndex(WORD index);
Returns the palette entry corresponding to the given index.

See also: *TDib::GetIndex*, *TDib::SetIndex*, *TDib::MapIndex*

GetBits

```
const void HUGE* GetBits() const;
void HUGE* GetBits();
```

Returns the *Bits* data member for this DIB.

See also: *TDib::Bits*

GetColor

```
TColor GetColor(int entry) const;
```

Returns the color for the given palette entry.

See also: *TDib::SetColor*, *TColor*, *TDib::FindColor*, *TDib::MapColor*

GetColors

```
inline const TRgbQuad far* GetColors() const;
inline TRgbQuad far* GetColors();
```

Returns the *bmiColors* value of this DIB.

See also: *TRgbQuad*

GetIndex

```
WORD GetIndex(int entry) const;
```

Returns the color index for the given palette entry.

See also: *TDib::SetIndex*, *TDib::FindIndex*, *TDib::MapIndex*

GetIndices

```
inline const WORD far* GetIndices() const;
inline WORD far* GetIndices();
```

Returns the *bmiColors* indices of this DIB.

See also: *TDib::GetColors*

GetInfo

```
inline const BITMAPINFO far* GetInfo() const;
inline BITMAPINFO far* GetInfo();
```

Returns this DIB's *Info* field.

See also: *TDib::Info*, *TDib::GetInfoHeader*, *BITMAPINFO*

GetInfoHeader

```
inline const BITMAPINFOHEADER far* GetInfoHeader() const;
inline BITMAPINFOHEADER far* GetInfoHeader();
```

Returns this DIB's *bmiHeader*.

See also: *TDib::Info*, *TDib::GetInfo*, *BITMAPINFOHEADER*

Height

```
inline int Height() const;
```

Returns 0 if *Info* is 0; otherwise returns *H*, this DIB's height.

See also: *TDib::Info*

IsOK

```
inline BOOL IsOK() const;
```

Returns FALSE if *Info* is 0, otherwise returns TRUE. If there is a problem with the construction of the DIB, memory is freed and *Info* is set to 0. Therefore, using *Info* is a reliable way to determine if the DIB is constructed correctly.

See also: *TDib* constructors, *TDib::Info*

IsPM

```
inline BOOL IsPM() const;
```

Returns TRUE if *IsCore* is TRUE; that is, if the DIB is an old-style PM DIB using core headers. Otherwise returns FALSE.

See also: *TDib::IsCore*

MapColor

```
int MapColor(TColor fromColor, TColor toColor, BOOL doAll = FALSE);
```

Maps the *fromColor* to the *toColor* in the current palette of this DIB. Returns the palette entry for the given color. Returns the palette entry for the *toColor* argument.

See also: *TDib::GetColor*, *TColor*, *TDib::SetColor*, *TDib::FindColor*

MapIndex

```
int MapIndex(WORD fromIndex, Word toIndex, BOOL doAll = FALSE);
```

Maps the *fromIndex* to the *toIndex* in the current palette of this DIB. Returns the palette entry for the *toIndex* argument.

See also: *TDib::FindIndex*, *TDib::SetIndex*, *TDib::GetIndex*

NumColors

```
inline long NumColors() const;
```

Returns 0 if *Info* is 0; otherwise returns *NumClrs*, the number of colors in this DIB's palette.

See also: *TDib::Info*

NumScans

```
inline int NumScans() const;
```

Returns 0 if *Info* is 0; otherwise returns *H*, the number of scans in this DIB.

See also: *TDib::StartScan*

operator <<

```
inline TClipboard& operator<<(TClipboard& clipboard, TDib& dib);
```

Writes the given *dib* to the given *clipboard*. Returns a reference to the resulting Clipboard, allowing the normal chaining of <<.

See also: *TClipboard*

```
operator BITMAPINFO() operator const BITMAPINFO far*() const;
operator BITMAPINFO far*();
```

Typecasts this DIB by returning a pointer to its bitmap information structure.

See also: *TDib::GetInfo*, *BITMAPINFO*

operator BITMAPINFOHEADER() operator const BITMAPINFOHEADER far*() const;
operator BITMAPINFOHEADER far*();

Typecasts this DIB by returning a pointer to its bitmap info header.

See also: *TDib::GetInfoHeader*

operator HANDLE() inline operator HANDLE() const;

Typecasts this DIB by returning its *Handle*.

See also: *TGdiObject::Handle*

operator TRgbQuad() operator const TRgbQuad far*() const;
operator TRgbQuad far*() const;

Typecasts this DIB by returning a pointer to its colors structure.

See also: *TDib::GetColors*, *TRgbQuad*

SetColor void SetColor(int entry, TColor color);

Sets the given color for the given palette entry.

See also: *TDib::GetColor*, *TColor*, *TDib::MapColor*, *TDib::FindColor*

SetIndex void SetIndex(int entry, WORD index);

Sets the given index for the given entry.

See also: *TDib::GetIndex*, *TDib::FindIndex*, *TDib::MapIndex*

Size inline TSize Size() const;

Returns *TSize(0,0)* if *Info* is 0; otherwise returns *TSize(W,H)*, the size of this DIB.

See also: *TDib::Info*, *TSize*

StartScan inline int StartScan() const;

Returns the DIB's starting scan line.

See also: *TDib::numScans*

ToClipboard void ToClipboard(TClipboard& clipboard);

Puts this DIB onto the specified Clipboard.

See also: *TClipboard*

Usage inline WORD Usage() const;

Returns the *Mode* for this DIB. This value tells *GDI* how to treat the color table.

See also: *TDib::Mode*

Width

```
inline int Width() const;
```

Returns 0 if *Info* is 0; otherwise returns *W*, the DIB width.

See also: *TDib::Info*

WriteFile

```
BOOL WriteFile(const char* filename);
```

Returns TRUE if the call is successful; otherwise returns FALSE.

Protected member functions

InfoFromHandle

```
void InfoFromHandle();
```

Locks this DIB's handle and extracts the remaining data member values from the DIB header.

See also: *TDib::GetInfoHeader*

LoadFile

```
BOOL LoadFile(const char* name);
```

Loads this DIB from the given file name. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *TDib::LoadResource*, *TDib* constructors

LoadResource

```
BOOL LoadResource(HINSTANCE instance, TResID resID);
```

Loads this DIB from the given resource and returns TRUE if successful.

See also: *TDib::LoadFile*, *TDib* constructors

Read

```
BOOL Read(TFile& file, long offBits = 0);
```

Reads data to this DIB, starting at offset *offBits*, from any file, BMP, or resource. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *TDib::LoadFile*

TDibDC class

dc.h

A DC class derived from *TDC*, *TDibDC* provides access to device-independent bitmaps (DIBs) using the DIB.DRV driver.

Public constructors

Constructor

```
TDibDC(const TDib& dib);
```


Creates a *TDibDC* object with the data provided by the given *TDib* object.

See also: *class TDib, TDC::TDC*

Constructor

```
TDibDC(const TDC& DC);
```

Creates a *TDib* object compatible with the given DC argument.

See also: *TDC::TDC*

TDocManager class

docmanag.h

TDocManager creates a document manager object that manages the list of current documents and registered templates, handles standard file menu commands, and displays the user-interface for file and view selection boxes. To provide support for documents and views, an instance of *TDocManager* must be created by the application and attached to the application.

The document manager normally handles events on behalf of the documents by using a response table to process the standard *CM_FILENEW*, *CM_FILEOPEN*, *CM_FILECLOSE*, *CM_FILESAVE*, *CM_FILESAVEAS*, and *CMVIEWCREATE* File menu commands. In response to a *CM_FILENEW* or a *CM_FILEOPEN* command, the document manager creates the appropriate document based on the user's selections. In response to the other commands, the document manager determines which of the open documents contains the view associated with the window that has focus. The menu commands are first sent to the window that is in focus and then through the parent window chain to the main window and finally to the application, which forwards the commands to the document manager.

When you create a *TDocManager* or a derived class, you must specify that it has either a multi-document (*dmMDI*) or single-document (*dmSDI*) interface. In addition, if you want the document manager to handle the standard file commands, you must OR these with *dmMenu*.

You can also enable or disable the document manager menu options by passing *dmSaveEnable* or *dmNoRevert* in the constructor. If you want to enable the File | Save menu option if the document is unmodified, pass the *dmSaveEnable* flag in the constructor. To disable the "Revert to Saved" menu option, pass *dmNoRevert* in the constructor.

When the application directly creates a new document and view, it can attach the view to its frame window, create MDI children, float the window, or create a splitter. However, when the document manager creates

a new document and view from the File | Open or File | New menu selection, the application doesn't control the process. To give the application control, the document manager sends messages after the new document and view are successfully created. Then, the application can use the information contained in the template to determine how to install the new document or view object.

TDocManager and derived classes can be created to support specialized needs such as an OLE 2.0 server.

Public data members

DocList

```
TDocument::List DocList;
```

Holds the list of attached documents or 0 if no documents exist.

Public constructors and destructor

Constructor

```
TDocManager(int mode);
```

Constructs a *TDocManager* object that supports either single (SDI) or multiple (MDI) open documents depending on the application. *mode* is set to either *dmMenu*, *dmMDI*, *dmSDI*, *dmSaveEnable*, or *dmNoRevert*. To install the standard *TDocManager* File menu commands, you must OR *dmMDI* or *dmSDI* with *dmMenu*. For example,

```
DocManager = new TDocManager(DocMode | dmMenu);
```

The document manager can then use its menu and response table to handle these events. If you do not specify the *dmMenu* parameter, you must provide the menu and functions to handle these commands. However, you can still use your application object's *DocManager* data member to access the document manager's functions.

See also: *dmxxxx* document manager mode constants

Destructor

```
virtual ~TDocManager();
```

Destroys a *TDocManager* object and removes attached documents and templates. The constructor resets *TDocTemplate::StaticHead* so that it points to the head of the static template list.

Public member functions

CmFileClose

```
virtual void CmFileClose();
```

Responds to a file close message. Tests to see if the document has been changed since it was last saved, and if not, prompts the user to confirm the save operation.

CmFileNew

```
virtual void CmFileNew();
```

Calls *CreateAnyDoc* and sets *dtNewDoc* to a value that indicates this is a new document with no path specified.

See also: *TDocManager::CreateAnyDoc*, *dtxxxx* document view constants

CmFileOpen

```
virtual void CmFileOpen();
```

Lets the user select a registered template from the list displayed in the dialog box. Calls *CreateAnyDoc*.

See also: *TDocManager::CreateAnyDoc*

CmFileRevert

```
virtual void CmFileRevert();
```

Reverts to the previously saved document.

CmFileSave

```
virtual void CmFileSave();
```

Responds to a file save message. Set *doc* to the current document. Calls *IsDirty* and returns *IDS_NOTCHANGED* if the document hasn't been changed since the last time it was saved.

CmFileSaveAs

```
virtual void CmFileSaveAs();
```

Prompts the user to enter a new name for the document.

CmViewCreate

```
virtual void CmViewCreate();
```

Responds to a view create message by creating a document view based on the specified directory path.

CreateAnyDoc

```
virtual TDocument* CreateAnyDoc(const char far* path, long flags= 0);
```

Creates a document based on the directory path and the specified template. *flags*, one of the document view constants, determines how the document template is created. If *path* is 0 and this is not a new document (the flag *dtNewDoc* is not set), it displays a dialog box. If *path* is 0, *dtNewDoc* is not set, and more than one template exists, it displays a dialog box and a list of templates.

See also: *TDocTemplate::CreateDoc*. *dtxxxx* document view constants

CreateAnyView

```
virtual TView* CreateAnyView(TDocument& doc, long flags= 0);
```

Creates a document view based on the directory path and specified template. *flags*, one of the document view constants, determines how the document template is created.

See also: *TDocTemplate::CreateView*, *dtxxxx* document view constants

DeleteTemplate

```
void DeleteTemplate(TDocTemplate&);
```

Removes a template from the list of templates attached to the document.

See also: *TDocManager::RefTemplate*

EvCanClose

```
BOOL EvCanClose();
```

Checks to see if all child documents can be closed before closing the current document. If any child returns FALSE, returns FALSE and aborts the process. If all children return TRUE, *EvCanClose* calls *TDocManager::FlushDoc* for each document. If *FlushDoc* finds that the document is dirty, it displays a message asking the user to save the document, discard any changes, or cancel the operation. If the document is not dirty and *EvCanClose* returns TRUE, *EvCanClose* returns TRUE.

See also: *TApplication::CanClose*, *TDocManager::FlushDoc*

EvPreProcessMenu

```
void EvPreProcessMenu(HMENU hmenu);
```

Called from *MainWindow*, *EvPreProcessMenu* loads and deletes a menu at the position specified by MF_POSITION or MF_POPUP. Your application can call *EvPreProcessMenu* to process the main window's menu before it is displayed.

See also: *TApplication::PreProcessMenu*

FlushDoc

```
virtual BOOL FlushDoc(TDocument& doc);
```

Updates the document with any changes and prompts the user for confirmation of updates.

GetApplication

```
inline TApplication* GetApplication();
```

Returns the current application.

GetCurrentDoc

```
virtual TDocument* GetCurrentDoc();
```

Calls *TWindow::GetFocus* to determine the window with the focus. Searches the list of documents and returns the document that contains the view with the focus. Returns 0 if no document has a view with focus.

MatchTemplate

```
TDocTemplate* MatchTemplate(const char far* path);
```

MatchTemplate returns the first registered template whose pattern matches the given file name. If no template is compatible with the supplied file name, it returns 0.

PostDocError

```
virtual UINT PostDocError(TDocument& doc, UINT sid, UINT choice = MB_OK);
```

Displays a message box with the error message passed as a string resource ID in *sid*. By default, the message box contains either an OK push button or a question mark icon. If an error message can't be found, *PostDocError* displays a "Message not found" message. *choice* can be one or more of the Windows MB_XXXX style constants. See the Windows API online Help for a description of the values. This function can be overridden.

See also: *TDocument::PostError*

PostEvent

```
virtual void PostEvent(int id, TDocument& doc);
```

If the current document changes, calls *::SendMessage* and passes a WM_OWLDOCUMENT message to indicate a change in the status of the document.

PostEvent

```
virtual void PostEvent(int id, TView& view);
```

If the current view changes, calls *::SendMessage* and passes a WM_OWLVIEW message to indicate a change in the status of the view.

RefTemplate

```
void RefTemplate(TDocTemplate&);
```

Adds a template to the list of templates attached to the document.

See also: *TDocManager::UnRefTemplate*

SelectAnySave

```
virtual TDocTemplate* SelectAnySave(TDocument& doc, BOOL samedoc = TRUE);
```

Selects a registered template to save with this document.

UnRefTemplate

```
void UnRefTemplate(TDocTemplate&);
```

Removes a template to the list of templates attached to the document.

See also: *TDocManager::RefTemplate*

Protected member functions

SelectDocPath

```
virtual int SelectDocPath(TDocTemplate** tpllist, int tplcount, char far* path, int buflen, long flags, BOOL save=FALSE);
```

Prompts the user to select one of the templates to use for the file to be opened. Returns the template index used for the selection or 0 if unsuccessful. For a file open operation, *save* is FALSE. For a file save operation, *save* is TRUE. This function can be overridden to provide a customized user-interface.

SelectDocType

```
virtual int SelectDocType(TDocTemplate** tpllist, int tplcount);
```

SelectDocType, which can be overridden, lets the user select a document type from a list of document templates. Returns the template index used for the selection or 0 if unsuccessful.

SelectViewType

```
virtual int SelectViewType(TDocTemplate** tpllist, int tplcount);
```

SelectViewType, which can be overridden, lets the user select a view name for a new view from a list of view names. Returns the template index used for the selection or 0 if unsuccessful.

Response table entries

Response table entry	Member function
EV_COMMAND(CM_FILECLOSE, CmFileClose)	CmFileClose
EV_COMMAND(CM_FILENEW, CmFileNew),	CmFileNew
EV_COMMAND(CM_FILEOPEN, CmFileOpen)	CmFileOpen
EV_COMMAND(CM_FILEREVERT, CmFileRevert)	CmFileRevert
EV_COMMAND(CM_FILESAVE, CmFileSave)	CmFileSave
EV_COMMAND(CM_FILESAVEAS, CmFileSaveAs)	CmFileSaveAs
EV_COMMAND(CM_VIEWCREATE, CmViewCreate)	CmViewCreate
EV_COMMAND_ENABLE(CM_FILECLOSE, CmEnableClose)	CmEnableClose
EV_COMMAND_ENABLE(CM_FILENEW, CmEnableNew)	CmEnableNew
EV_COMMAND_ENABLE(CM_FILEOPEN, CmEnableOpen)	CmEnableOpen
EV_COMMAND_ENABLE(CM_FILEREVERT, CmEnableRevert)	EmEnableRevert
EV_COMMAND_ENABLE(CM_FILESAVE, CmEnableSave)	CmEnableSave
EV_COMMAND_ENABLE(CM_FILESAVEAS, CmEnableSaveAs)	CmEnableSaveAs
EV_COMMAND_ENABLE(CM_VIEWCREATE, CmEnableCreate)	CmEnableCreate
EV_WM_OWLCANCLOSE	EvCanClose
EV_WM_OWLPREPROCMENU	EvPreProcessMenu

TDocTemplate class**docmanag.h**

TDocTemplate is an abstract base class that contains document template functionality. *TDocTemplate* classes create documents and views from resources and handle document naming and browsing. The document manager maintains a list of the current template objects. Each document type requires a separate document template.

Public member functions**ClearFlag**

```
inline void ClearFlag(long flag);
```

Clears a document view constant.

See also: *dtxxxx* document view constants

CreateDoc

```
virtual TDocument* CreateDoc(const char far* path, long flags = 0) = 0;
```

A pure virtual function that must be defined in a derived class, *CreateDoc* creates a document based on the directory path (*path*) and the specified template and *flags* value. If the *path* is 0 and the new flag (*dtNewDoc*) is not set, the dialog box is displayed.

See also: *TDocManager::CreateAnyDoc*

CreateView

```
virtual TView* CreateView(TDocument& doc, long flags) = 0;
```

A pure virtual function that must be defined in a derived class, *CreateView* creates the view specified by the document template class.

See also: *TDocManager::CreateAnyView*

GetDefaultExt

```
inline const char far* GetDefaultExt() const;
```

Gets the default extension to use if the user has entered the name of a file without any extension. If there is no default extension, *GetDefaultExt* contains 0.

GetDescription

```
inline const char far* GetDescription() const;
```

Gets the template description to put in the file-selection list box or the File | New menu-selection list box.

GetDirectory

```
inline const char far* GetDirectory() const;
```

Gets the directory path to use when searching for matching files. This will get updated if a file is selected and the *dtUpdateDir* flag is set.

See also: *dt* document view constants

GetDocManager

```
inline TDocManager* GetDocManager() const;
```

Points to the document manager.

GetFileFilter

```
inline const char far* GetFileFilter() const;
```

Gets any valid document matching pattern to use when searching for files.

GetFlags

```
inline long GetFlags() const;
```

Gets the document view constants, which indicate how the document is created and opened.

See also: *dtxxxx* document view constants

GetViewName

```
virtual const char far* GetViewName() = 0;
```

A pure virtual function that must be defined in a derived class, *GetViewName* gets the name of the view associated with the template.

- IsFlagSet** `inline BOOL IsFlagSet(long flag);`
Returns nonzero if the document view flag is set.
See also: *dtxxxx* document view constants
- IsMyKindOfDoc** `virtual TDocument* IsMyKindOfDoc(TDocument& doc)=0;`
A pure virtual function that must be defined in a derived class, *IsMyKindOfDoc* tests if the template belongs to the same class as the document or to a derived class.
See also: *TDocTemplateT::IsMyKindOfDoc*
- IsVisible** `inline BOOL IsVisible();`
Indicates whether the document can be displayed in the file selection dialog box. A document is visible if *dtHidden* isn't set and *Description* isn't 0.
- SelectSave** `BOOL SelectSave(TDocument& doc);`
Prompts the user to select a file name for the document. Filters out read-only files.
- SetDefaultExt** `void SetDefaultExt(const char far*);`
Sets the default extension to use if the user has entered the name of a file without any extension. If there is no default extension, *SetDefaultExt* contains 0.
- SetDirectory** `void SetDirectory(const char far*, int len);`
Sets the directory path to use when searching for matching files. This will get updated if a file is selected and the *dtUpdateDir* flag is set.
See also: *TDocTemplate::GetDirectory*
- SetDocManager** `inline void SetDocManager(TDocManager* dm);`
Sets the current document manager to the argument *dm*:
- SetFileFilter** `void SetFileFilter(const char far*);`
Sets the valid document matching pattern to use when searching for files.
- SetFlag** `inline void SetFlag(long flag);`
Sets the document view constants, which indicate how the document is created and opened.
See also: *dtxxxx* document view constants

Protected constructors and destructor

Constructor

```
TDocTemplate(const char far* desc, const char far* filt, const char
             far* dir, const char far* ext, long flags):
```

Constructs a *TDocTemplate* with the specified file description (*desc*), file filter pattern (*filt*), search path for viewing the directory (*dir*), default file extension (*ext*), and flags representing the view and creation options (*flags*). Sets *Description* to *desc*, *FileFilter* to *filt*, *Directory* to *dir*, *DefaultExt* to *ext*, and *Flags* to *flags*. Then, adds this template to the document manager's template list. If the document manager is not yet constructed, adds the template to a static list, which the document manager will later add to its template list.

Destructor

```
~TDocTemplate();
```

Destroys a *TDocTemplate* object and frees the data members (*FileFilter*, *Description*, *Directory*, and *DefaultExt*). The destructor is called only when no views or documents are associated with the template. Instead of calling this destructor directly, use the *Delete* member function.

Protected member functions

InitDoc

```
TDocument* InitDoc(TDocument* doc, const char far* path, long flags);
```

Called only from the subclass to continue *CreateDoc* processing.

See also: *TDocTemplate::CreateDoc*

InitView

```
TView* InitView(TView* view);
```

Called only from the subclass to continue *CreateView* processing.

See also: *TDocTemplate::CreateView*

TDocTemplateT<D,V> class

docmanag.h

To register the associated document and view classes, a parameterized subclass, *TDocTemplateT<D,V>*, is used to construct a particular document and view, where *D* represent the document class and *V* represents the view class. The parameterized template classes are created using a macro, which also generates the associated streamable support. The document and view classes are provided through the use of a parameterized subclass. The template class name is used as a **typedef** for the parameterized class. For example,

```
DEFINE_DOC_TEMPLATE_CLASS(TFileDocument, TEditView, MyEditFile)
```

You can instantiate a document template using either a static member or an explicit construction. For example,

```
MyEditFile et1("Edit text files",
               "*.txt", "D:\\doc", ".TXT", dtNoAutoView);
new MyEditFile(.....)
```

When a document template is created, the document manager (*TDocManager*) registers the template. When the document template's delete function is called to delete the template, it is no longer visible to the user. However, it remains in memory as long as any documents still use it.

Public constructors

Constructor

```
TDocTemplateT(const char far* filt, const char far* desc, const char far*
              dir, const char far* ext, long flags);
```

Constructs a *TDocTemplateT* with the specified file description (*desc*), file filter pattern (*filt*), search path for viewing the directory (*dir*), default file extension (*ext*), and flags representing the view and creation options (*flags*).

Public member functions

CreateDoc

```
D* CreateDoc(const char far* path, long flags);
```

CreateDoc creates a document of type *D* based on the directory path (*path*) and *flags* value.

See also: *TDocTemplate::CreateDoc*

CreateView

```
TView* CreateView(TDocument& doc, long flags);
```

CreateView creates the view specified by the document template class.

See also: *TDocManager::CreateAnyView*

IsMyKindOfDoc

```
D* IsMyKindOfDoc(TDocument& doc);
```

IsMyKindOfDoc tests to see if the document (*doc*) is the same class as the template's document class or if the document is a derived class. If the template can't use the document, *IsMyKindOfDoc* returns 0.

See also: *TDocTemplate::IsMyKindOfDoc*

IsMyKindOfView

```
V* IsMyKindOfView(TView& view);
```

IsMyKindOfView tests to see if the view (*view*) is the same class as the template's view class or if the view is a derived class. If the template can't use the view, *IsMyKindOfView* returns 0.

GetViewName inline virtual const char far* GetViewName();

GetViewName gets the name of the view associated with the template.

TDocument class

docview.h

TDocument is an abstract base class that serves as an interface between the document, its views, and the document manager (*TDocManager* class). *TDocument* creates, destroys, and sends messages about the view. For example, if the user changes a document, *TDocument* tells the view that the document has been updated.

In order to send messages to its associated views, the document maintains a list of all the views existing for that document and communicates with the views using ObjectWindows event-handling mechanism. Rather than using the function *SendMessage*, the document accesses the view's event table. The views can update the document's data by calling the member functions of the particular document. Views can also request streams, which are constructed by the document.

Both documents and views have lists of properties for their applications to use. When documents and views are created or destroyed, messages are sent to the application, which can then query the properties to determine how to process the document or view. It is the document manager's responsibility to determine if a particular view is appropriate for the given document.

Because the property attribute functions are virtual, a derived class (which is called first) might override the properties defined in a base class. Each derived class must implement its own property attribute types of either string or binary data. If the derived class duplicates the property names of the parent class, it should provide the same behavior and data type as the parent. See Chapter 9 in the *ObjectWindows Programmer's Guide* for more information about how to use property attributes.

Although documents are usually associated with files, they do not necessarily have to be files; they can also consist of database tables, mail systems, fax or modem transmissions, disk directories, and so on. See Chapter 9 in the *ObjectWindows Programmer's Guide* for a visual representation of the general form of communication among views (some of which contain windows), documents, and persistent (or disk) storage.

Public data members

ChildDoc

List ChildDoc;

The list of child documents associated with this document.

Property enum

```
enum {
    PrevProperty = 0,
    DocumentClass,
    TemplateName,
    ViewCount,
    StoragePath,
    DocTitle,
    NextProperty,
};
```

These property values, defined for *TDocument*, are available in classes derived from *TDocument*. *PrevProperty* and *NextProperty* are delimiters for every document's property list. See Chapter 9 in the *ObjectWindows Programmer's Guide* for information about how to use these enumerated property values.

Tag

LPVOID Tag;

Tag holds a pointer to the application-defined data. Typically, you can use *Tag* to install a pointer to your own application's associated data structure. *Tag*, which is initialized to 0 at the time a *TDocument* is constructed, is not used otherwise by the document view classes.

Public constructors and destructor

Constructor

TDocument(TDocument* parent = 0);

Although you don't create a *TDocument* object directly, you must call the constructor when you create a derived class. *parent* points to the parent of the new document. If no parent exists, *parent* is 0.

Destructor

virtual ~TDocument();

Deletes a *TDocument* object. Normally, *Close* is called first. The *~TDocument* destroys all children and closes all open streams. If this is the last document that used the template, closes the object's template, and any associated views, deletes the object's stream, and removes itself from the parent's list of children if a parent exists. If there is no parent, removes itself from the document manager's document list.

See also: *TDocument::Close*

Public member functions

CanClose

```
virtual BOOL CanClose();
```

Checks to see if all child documents can be closed before closing the current document. If any child returns `FALSE`, *CanClose* returns `FALSE` and aborts the process. If all children return `TRUE`, calls *TDocManager::FlushDoc*. If *FlushDoc* finds that the document has been changed but not saved, it displays a message asking the user to either save the document, discard any changes, or cancel the operation. If the document has not been changed and all children's *CanClose* functions return `TRUE`, this *CanClose* function returns `TRUE`.

See also: *TView::CanClose*, *TDocManager::FlushDoc*

Close

```
virtual BOOL Close();
```

Closes the document but does not delete or detach the document. Before closing the document, *Close* checks any child documents and tries to close them before closing the parent document. Even if you write your own *Close* function, call *TDocument's* version so that all child documents are checked before the parent document is closed.

Commit

```
virtual BOOL Commit(BOOL force = FALSE);
```

Saves the current data to storage. When a file is closed, the document manager calls either *Commit* or *Revert*. If *force* is `TRUE`, all data is written to storage. *TDocument's* *Commit* checks any child documents and commits their changes to storage also. Before the current data is saved, all child documents must return `TRUE`. If all child documents return `TRUE`, *Commit* flushes the views for operations that occurred since the last time the view was checked. Once all data for the document is updated and saved, *Commit* returns `TRUE`.

See also: *TDocument::Revert*

FindProperty

```
virtual int FindProperty(const char far* name);
```

FindProperty gets the property index, given the property name (*name*). Returns the integer index number that corresponds to the name or 0 if the name isn't found in the list of properties.

See also: *pfxxxx* property attribute constants, *TDocument::PropertyName*

GetDocManager

```
inline TDocManager& GetDocManager();
```

Returns a pointer to the current document manager.

GetDocPath

```
inline LPCSTR GetDocPath();
```

Returns the directory path for the document. This might change the SaveAs operation.

GetOpenMode

```
int GetOpenMode;
```

Gets the mode and protection flag values for the current document.

See also: *TDocument::SetOpenMode*

GetParentDoc

```
inline TDocument* GetParentDoc();
```

Returns the parent document of the current document or 0 if there is no parent document.

GetProperty

```
virtual int GetProperty(int index, void far* dest, int textlen=0);
```

Returns the total number of properties for this document where *index* is the property index, *dest* contains the property data, and *textlen* is the size of the array. If *textlen* is 0, property data is returned as binary data; otherwise, property data is returned as text data.

See also: *pfxxxx property attribute* constants, *TDocument::SetProperty*

GetTemplate

```
inline TDocTemplate* GetTemplate();
```

Gets the template used for document creation. The template can be changed during a SaveAs operation.

GetTitle

```
inline LPCSTR GetTitle();
```

Returns the title of the document.

HasFocus

```
BOOL HasFocus(HWND hwnd);
```

Used by the document manager to determine if the document contains a view with a focus.

InStream

```
inline virtual TInStream* InStream(int mode, LPCSTR strmId=0);
```

Generic input for the particular storage medium, *InStream* returns a pointer to a *TInStream*. *mode* is a combination of the ios bits defined in *iostream.h*. *strmId* is one of the *shxxxx* file-sharing modes. *strmId* is used for documents that support named streams. Override this function to provide streaming for your document class.

See also: *TDocument::OutStream*

IsDirty

```
virtual BOOL IsDirty();
```

Returns TRUE if the document or one of its views has changed but has not been saved.

IsOpen

```
inline virtual BOOL IsOpen();
```

Checks to see if the document has any streams in its stream list. Returns FALSE if no streams are open; otherwise, returns TRUE.

NextStream

```
TStream* NextStream(const TStream* strm);
```

Gets the next entry in the stream. Holds 0 if none exists.

NextView

```
TView* NextView(const TView* view);
```

Gets the next view in the list of views. Holds 0 if none exists.

NotifyViews

```
BOOL NotifyViews(int event, long item=0, TView* exclude=0);
```

Notifies the views of the current document and the views of any child documents of a change. In contrast to *QueryViews*, *NotifyViews* sends notification of an event to all views and returns TRUE if any views returned a TRUE result. The event, EV_OWLNOTIFY, is sent with an event code, which is private to the particular document and view class, and a **long** argument, which can be cast appropriately to the actual type passed in the argument of the response function.

See also: *TDocument::QueryViews*

Open

```
inline virtual BOOL Open(int mode, LPCSTR path = 0);
```

Opens the document using the path specified by *DocPath*. Sets *OpenMode* to *mode*. *TDocument*'s version always returns TRUE and actually performs no actions. Other classes override this function to open specified file documents and views.

See also: *TFileDocument::Open*

OutStream

```
inline virtual TOutStream* OutStream(int mode, LPCSTR strmId = 0);
```

Generic output for the particular storage medium, *OutStream* returns a pointer to a *TOutStream*. *mode* is a combination of the ios bits defined in *iostream.h*. *strmId* is used for documents that support named streams. *TDocument*'s version always returns 0. Override this function to provide streaming for your document class.

See also: *TDocument::InStream*

PostError

```
virtual UINT PostError(UINT sid, UINT choice = MB_OK);
```

Posts the error message passed as a string resource ID in *sid*. *choice* is one or more of the Windows MB_Xxxx style constants. See the Windows API online Help for a description of the values.

See also: *TDocManager::PostDocError*

PropertyCount

```
inline virtual int PropertyCount();
```

Gets the total number of properties for the *TDocument* object. Returns *NextProperty* -1.

See also: *pfxxxx* property attribute constants

PropertyFlags

```
virtual int PropertyFlags(int index);
```

Returns the attributes of a specified property given the index (*index*) of the property whose attributes you want to retrieve.

See also: *pfxxxx* property attribute constants, *TDocument::FindProperty*, *TDocument::PropertyName*

PropertyName

```
virtual const char* PropertyName(int index);
```

Returns the name of the property given the index value (*index*).

See also: *pfxxxx* property attribute constants, *TDocument::FindProperty*

QueryViews

```
TView* QueryViews(int event, long item=0, TView* exclude=0);
```

Queries the views of the current document and the views of any child documents about a specified event, but stops at the first view that returns TRUE. In contrast to *NotifyViews*, *QueryViews* returns a pointer to the first view that responded to an event with a TRUE result. The event, *EV_OWLNOTIFY*, is sent with an event code (which is private to the particular document and view class) and a **long** argument (which can be cast appropriately to the actual type passed in the argument of the response function).

See also: *TDocument::NotifyViews*

Revert

```
virtual BOOL Revert( BOOL clear = FALSE);
```

Performs the reverse of *Commit* and cancels any changes made to the document since the last *commit*. If *clear* is TRUE, data is not reloaded for views. *Revert* also checks all child documents and cancels any changes if all children return TRUE. When a file is closed, the document manager calls either *Commit* or *Revert*. Returns TRUE if the operation is successful.

See also: *TDocument::Commit*

RootDocument

```
virtual TDocument& RootDocument();
```

Returns the **this** pointer as the root document.

SetDocManager

```
inline void SetDocManager(TDocManager& dm);
```

Sets the current document manager to the argument *dm*.

SetDocPath

```
virtual BOOL SetDocPath(LPCSTR path);
```

Sets the document path for Open and Save operations.

- SetOpenMode** `void SetOpenMode(int mode);`
Sets the mode and protection flag values for the current document.
See also: *TDocument::GetOpenMode*
- SetProperty** `virtual BOOL SetProperty(int index, const void far* src);`
Sets the value of the property, given the index of the property, and *src*, the data type (either binary or text) to which the property must be set.
See also: *pfxxxx* property attribute constants, *TDocument::GetProperty*
- SetTemplate** `BOOL SetTemplate(TDocTemplate* tpl);`
Sets the document template. However, if the template type is incompatible with the file, the document manager will refuse to save the file as this template type.
- SetTitle** `virtual void SetTitle(LPCSTR title);`
Sets the title of the document.

Protected data members

- DirtyFlag** `BOOL DirtyFlag;`
Indicates that unsaved changes have been made to the document. Views can also independently maintain their local disk status.

Protected member functions

- AttachStream** `virtual void AttachStream(TStream& strm);`
Called from *TStream*'s constructor, *AttachStream* attaches a stream to the current document.
- DetachStream** `virtual void DetachStream(TStream& strm);`
Called from *TStream*'s destructor, *AttachStream* detaches a stream associated with the current document.

TDocument::List class**docview.h**

The *TDocument::List* nested class encapsulates the chain of documents. It allows addition, removal, and destruction of documents from the document list.

Public constructors and destructor

Constructor	<code>List();</code> Constructs a <i>TDocument::List</i> object.
Destructor	<code>inline ~List();</code> Destroys a <i>TDocument::List</i> object.

Public member functions

Destroy	<code>void Destroy();</code> Deletes all documents.
Insert	<code>BOOL Insert(TDocument* doc);</code> Inserts a new document into the document list. Fails if the document already exists.
Remove	<code>BOOL Remove(TDocument* doc);</code> Removes a document from the document list.

TDropInfo class**point.h**

TDropInfo is a simple class that supports file-name drag and drop operations using the Windows API `WM_DROPFILES` message. Each *TDropInfo* object has a private handle to the `HDROP` structure returned by the `WM_DOPFILES` message. Member functions simplify calls to `::DragQueryFile`, `::DragQueryPoint`, and `::DragFinish`.

Public constructors

Constructor	<code>TDropInfo(HDROP handle);</code> Creates a <i>TDropInfo</i> object with <i>Handle</i> set to the given <i>handle</i> .
--------------------	--

See also: *TDropInfo::Handle*

Public member functions

DragFinish

```
inline void DragFinish();
```

Releases any memory allocated for the transferring of this *TDropInfo* object's files during drag operations.

See also: *::DragFinish*

DragQueryFile

```
inline UINT DragQueryFile( UINT index, char far* name, UINT nameLen)
```

Retrieves the name of the file and related information for this *TDropInfo* object. If *index* is set to -1 (0xFFFF), *DragQueryFile* returns the number of dropped files. This is equivalent to calling *DragQueryFileCount*.

If *index* lies between 0 and the total number of dropped files for this object, *DragQueryFile* copies to the *name* buffer (of length *nameLen* bytes) the name of the dropped file that corresponds to *index*, and returns the number of bytes actually copied.

If *name* is 0, *DragQueryFile* returns the required buffer size (in bytes) for the given *index*. This is equivalent to calling *DragQueryFileNameLen*.

See also: *::DragQueryFile*, *TDropInfo::DragQueryPoint*, *TDropInfo::DragQueryFileCount*, *TDropInfo::DragQueryFileCount*

DragQueryFileCount

```
inline UINT DragQueryFileCount();
```

Returns the number of dropped files in this *TDropInfo* object. This call is equivalent to calling *DragQueryFile(-1, 0, 0)*.

See also: *TDropInfo::DragQueryFile*

DragQueryFileNameLen

```
inline UINT DragQueryFileNameLen(UINT index)
```

Returns the length of the name of the file in this *TDropInfo* object corresponding to the given index. This call is equivalent to calling *DragQueryFile(index, 0, 0)*.

See also: *TDropInfo::DragQueryFile*

DragQueryPoint

```
inline BOOL DragQueryPoint(TPoint& point)
```

Retrieves the mouse pointer position when this object's files are dropped and copies the coordinates to the given *point* object. *point* refers to the window that received the WM_DROPFILES message. *DragQueryPoint* returns TRUE if the drop occurs inside the window's client area, otherwise FALSE.

See also: `::DragQueryPoint`, `TPoint`, `WM_DROPFILES`

operator HDROP()

inline operator HDROP();

Typecasting operator that returns *Handle*.

See also: `TDropInfo::Handle`

TEdgeConstraint struct

layoutco.h

TEdgeConstraint adds member functions that set edge (but not size) constraints. *TEdgeConstraint* always places your window one pixel above the other window and then adds margins. For example, if the margin is 4, *TEdgeConstraint* places your window 5 pixels above the other window. The margin, which does not need to be measured in pixels, is defined using the units specified by the constraint. Therefore, if the margin is specified as 8 layout units (which are then converted to 12 pixels), your window would be placed 13 pixels above the other window. See *TLayoutWindow* for an example of layout constraints.

See also: *TLayoutConstraint*

Public member functions

Above

inline void Above (TWindow *sibling, int margin = 0)

Positions your window above a sibling window. You must specify the sibling window and an optional margin between the two windows. If no margin is specified, *Above* sets the bottom of one window one pixel above the top of the other window.

See also: *TEdgeConstraint::Below*

Absolute

inline void Absolute (TEdge edge, int value)

Sets an edge of your window to a fixed value.

See also: *TEdgeConstraint::PercentOf*

Below

inline void Below(TWindow *sibling, int margin = 0);

Positions your window with respect to a sibling window. You must specify the sibling window and an optional margin between the two windows. If no margin is specified, *Below* sets the top of one window one pixel below the bottom of the other window.

See also: *TEdgeConstraint::Above*

LeftOf

```
inline void LeftOf (TWindow *sibling, int margin = 0)
```

Positions one window with respect to a sibling window. You can specify the sibling window and an optional margin between the two windows.

See also: *TEdgeConstraint::RightOf*

PercentOf

```
inline void PercentOf (TWindow *otherWin, TEdge edge, int percent)
```

Specifies that the edge of one window indicated in *edge* should be a percentage of the corresponding edge of another window or *otherWin*.

RightOf

```
inline void RightOf (TWindow *sibling, int margin = 0)
```

Positions one window with respect to a sibling window. You can specify the sibling window and an optional margin between the two windows.

See also: *TEdgeConstraint::LeftOf*

SameAs

```
inline void SameAs (TWindow *otherWin, TEdge edge)
```

Sets the edge of your window indicated by *edge* equivalent to the corresponding edge of the window in *otherWin*.

See also: *TEdgeConstraint::Set*

Set

```
inline void Set(TEdge edge, TRelationship rel, TWindow *otherWin,
               TEdge otherEdge, int value = 0);
```

Used for setting arbitrary edge constraints, *Set* specifies that your window's edge should be of a specified relationship to *otherWin*'s specified edge.

See also: *TEdgeConstraint::SameAs*

TEdgeOrSizeConstraint struct**layoutco.h**

Derived from *TEdgeConstraint*, *TEdgeOrSizeConstraint* is a template class that supports size constraints in addition to all the operations that *TEdgeConstraint* provides. The width or height is specified in the template instantiation of this class. There are two versions of each member function: one sets both edge and size constraints; the other sets only edge constraints.

Public member functions**Absolute**

```
inline void Absolute (int value)
```

Sets the width or height of your window to a fixed value.

Absolute

```
inline void Absolute (TEdge edge, int value)
```

Used to determine edge constraints only, *Absolute* sets the edge of your window to a fixed value.

See also: *TEdgeConstraint::Absolute*

PercentOf

```
inline void PercentOf (TWindow *otherWin, int percent,
                      TWidthHeight otherWidthHeight = widthOrHeight)
```

Although a window's width or height defaults to being a percentage of the sibling or parent window's corresponding dimension, it can also be a percentage of the sibling or parent's opposite dimension. For example, one window's width can be 50% of the parent window's height.

See also: *TEdgeOrSizeConstraint::Absolute*

PercentOf

```
inline void PercentOf (TWindow *otherWin, TEdge edge, int percent)
```

Used to determine edge constraints only, *PercentOf* specifies that the edge of one window indicated in *edge* should be a percentage of the corresponding edge of another window or *otherWin*.

See also: *TEdgeConstraint::PercentOf*

SameAs

```
inline void SameAs (TWindow *otherWin,
                   TWidthHeight otherWidthHeight = widthOrHeight,
                   int value = 0)
```

Although a window's width or height defaults to being the same as the sibling or parent window's corresponding dimension, it can be the same of the sibling's or parent's opposite dimension. For example, one window's width can be the same as the parent window's height.

See also: *TEdgeOrSizeConstraint::PercentOf*

SameAs

```
inline void SameAs (TWindow *otherWin, TEdge edge)
```

Used to determine edge constraints only, *SameAs* sets the edge of one window the same as the corresponding edge of the other window specified in *otherWin*.

See also: *TEdgeConstraint::SameAs*

TEdit class**edit.h**

A *TEdit* is an interface object that represents an edit control interface element in Windows. A *TEdit* object must be used to create an edit control in a parent *TWindow*. A *TEdit* can be used to facilitate communication

between your application and the edit controls of a *TDialog*. This class is streamable.

There are two styles of edit control objects: single line and multiline. Multiline edit controls allow editing of multiple lines of text.

The position of the first character in an edit control is zero. For a multiline edit control, the position numbers continue sequentially from line to line; line breaks count as two characters.

Most of *TEdit*'s member functions manage the edit control's text. *TEdit* also includes some automatic member response functions that respond to selections from the edit control's parent window menu, including cut, copy, and paste. Two important member functions inherited from *TEdit*'s base class (*TStatic*) are *GetText* and *SetText*.

Public constructors

Constructor

```
TEdit(TWindow* parent, int Id, const char far *text, int x, int y, int w,
      int h, UINT textLen, BOOL multiline = FALSE, TModule* module = 0);
```

Constructs an edit control object with a parent window (*parent*). Sets the creation attributes of the edit control and fills its *Attr* data members with the specified control ID (*Id*), position (*x, y*) relative to the origin of the parent window's client area, width (*w*), and height (*h*).

If text buffer length (*textLen*) is 0 or 1, there is no explicit limit to the number of characters that can be entered. Otherwise *textLen - 1* characters can be entered. By default, initial text (*text*) in the edit control is left-justified and the edit control has a border. Multiline edit controls have horizontal and vertical scroll bars.

Constructor

```
TEdit(TWindow* parent, int resourceID, UINT textLen, TModule* module = 0);
```

Constructs a *TEdit* object to be associated with an edit control of a *TDialog*. Invokes the *TStatic* constructor with identical parameters. The *resourceID* parameter must correspond to an edit resource that you define. Enables the data transfer mechanism by calling *EnableTransfer*.

See also: *TStatic::TStatic*

Public member functions

CanUndo

```
BOOL CanUndo();
```

Returns TRUE if it is possible to undo the last edit.

See also: *TEdit::Undo*

ClearModify

```
void ClearModify();
```

Resets the change flag of the edit control causing *IsModified* to return FALSE. The flag is set when text is modified.

See also: *TEdit::IsModified*

Copy

```
void Copy();
```

Copies the currently selected text into the Clipboard.

Cut

```
void Cut();
```

Deletes the currently selected text and copies it into the Clipboard.

DeleteLine

```
BOOL DeleteLine(int lineNumber);
```

Deletes the text in the line specified by *lineNumber* in a multiline edit control. If -1 passed, deletes the current line. *DeleteLine* does not delete the line break and affects no other lines. Returns TRUE if successful. Returns FALSE if *lineNumber* is not -1 and is out of range or if an error occurs.

DeleteSelection

```
BOOL DeleteSelection();
```

Deletes the currently selected text, and returns FALSE if no text is selected.

DeleteSubText

```
BOOL DeleteSubText(UINT startPos, UINT endPos);
```

Deletes the text between the starting and ending positions specified by *startPos* and *endPos*, respectively. *DeleteSubText* returns TRUE if successful.

EmptyUndoBuffer

```
void EmptyUndoBuffer();
```

If an operation inside the edit control can be undone, the edit control undo flag is set. *EmptyUndoBuffer* resets or clears this flag.

FormatLines

```
inline void FormatLines(BOOL addEOL);
```

Indicates if the end-of-line characters (carriage return, linefeed) are to be added or removed from text lines that are wordwrapped in a multiline edit control. Returns TRUE if these characters are placed at the end of wordwrapped lines or FALSE if they are removed.

GetFirstVisibleLine

```
inline int GetFirstVisibleLine() const;
```

Indicates the topmost visible line in an edit control. For single-line edit controls, the return value is 0. For multiline edit controls, the return value is the index of the topmost visible line.

GetHandle

```
inline HANDLE GetHandle() const;
```


Returns the data handle of the buffer that holds the contents of the control window.

See also: *:TEdit::SetHandle*

GetLine

```
BOOL GetLine(char far* str, int strSize, int lineNumber);
```

Retrieves a line of text (whose line number is supplied) from the edit control and returns it in *str* (NULL-terminated). *strSize* indicates how many characters to retrieve. *GetLine* returns FALSE if it is unable to retrieve the text or if the supplied buffer is too small.

See also: *TStatic::GetText*, *TEdit::GetNumLines*, *TEdit::GetLineLength*

GetLineFromPos

```
inline int GetLineFromPos(UINT charPos);
```

From a multiline edit control, returns the line number on which the character position specified by *charPos* occurs. If *charPos* is greater than the position of the last character, the number of the last line is returned. If *charPos* is -1, the number of the line that contains the first selected character is returned. If there is no selection, the line containing the caret is returned.

See also: *::EM_LINEFROMCHAR*

GetLineIndex

```
inline UINT GetLineIndex(int lineNumber);
```

In a multiline edit control, *GetLineIndex* returns the number of characters that appear before the line number specified by *lineNumber*. If *lineNumber* is -1, *GetLineIndex* returns the number of the line that contains the caret is returned.

See also: *::EM_LINEINDEX*

GetLineLength

```
int GetLineLength(int lineNumber);
```

From a multiline edit control, *GetLineLength* returns the number of characters in the line specified by *lineNumber*. If it is -1, the following applies: if no text is selected, *GetLineLength* returns the length of the line where the caret is positioned; if text is selected on the line, *GetLineLength* returns the line length minus the number of selected characters; if selected text spans more than one line, *GetLineLength* returns the length of the lines minus the number of selected characters.

GetNumLines

```
inline int GetNumLines();
```

Returns the number of lines that have been entered in a multiline edit control: 1 if the edit control has no text (if it has one line with no text in it), or 0 if there is no text or if an error occurs.

See also: *::EM_GETNUMLINES*

GetPasswordChar

```
inline UINT GetPasswordChar() const;
```

Returns the character to be displayed in place of a user-typed character. When the edit control is created with the `ES_PASSWORD` style specified, the default display character is an asterisk (*).

See also: `TEdit::SetPasswordChar`, `::EM_GETPASSWORDCHAR`, `::ES_PASSWORD`

GetRect

```
inline void GetRect(TRect& frmtRect) const;
```

Gets the formatting rectangle of a multiline edit control.

See also: `TEdit::SetRect`, `TEdit::SetRectNP`, `::EM_GETRECT`

GetSelection

```
inline void GetSelection(UINT& startPos, UINT& endPos);
```

Returns the starting (*startPos*) and ending (*endPos*) positions of the currently selected text. By using `GetSelection` in conjunction with `GetSubText`, you can get the currently selected text.

See also: `TEdit::GetSubText`

GetSubText

```
void GetSubText(char far* str, UINT startPos, UINT endPos);
```

Retrieves the text in an edit control from indices *startPos* to *endPos* and returns it in *str*.

See also: `TEdit::GetSelection`

GetWordBreakProc

```
inline EDITWORDBREAKPROC GetWordBreakProc() const;
```

Retrieves the current wordwrap function. Returns the address of the wordwrap function defined by the application or 0 if none exists.

See also: `TEdit::SetWordBreakProc`, `::EM_GETWORDBREAKPROC`

Insert

```
inline void Insert(const char far* str);
```

Inserts the text supplied in *str* into the edit control at the current text insertion point (cursor position), and replaces any currently selected text. `Insert` is similar to `Paste`, but does not affect the Clipboard.

See also: `TEdit::Paste`

IsModified

```
BOOL IsModified();
```

Returns TRUE if the user has changed the text in the edit control.

See also: `TEdit::ClearModify`

LockBuffer

```
char far* LockBuffer(UINT newsize=0);
```

Locks the edit control's buffer and returns a pointer to the buffer. Passing *newsize* greater than 0 causes the buffer to be resized to *newsize*. You must call `Unlock` when you are finished.

See also: *TEdit::UnLockBuffer*

Paste

```
void Paste();
```

Inserts text from the Clipboard into the edit control at the current text insertion point (cursor position).

See also: *TEdit::CMEditPaste*

Scroll

```
inline void Scroll(int horizontalUnit, int verticalUnit);
```

Scrolls a multiline edit control horizontally and vertically using the numbers of characters specified in *horizontalUnit* and *verticalUnit*. Positive values result in scrolling to the right or down in the edit control, and negative values result in scrolling to the left or up.

Search

```
int Search(UINT startPos, const char far* text, BOOL caseSensitive=FALSE,
           BOOL wholeWord=FALSE, BOOL up=FALSE);
```

Performs either a case-sensitive or case-insensitive search for the supplied text. If the text is found, the matching text is selected, and *Search* returns the position of the beginning of the matched text. If the text is not found in the edit control's text, *Search* returns -1. If -1 is passed as *startPos*, then the search starts from either the end or the beginning of the currently selected text, depending on the search direction.

SetHandle

```
inline void SetHandle(HLOCAL localMem);
```

Sets a handle to the text buffer used to hold the contents of a multiline edit control.

See also: *:TEdit::GetHandle*

SetPasswordChar

```
void SetPasswordChar(UINT ch);
```

SetPasswordChar sets the character to be displayed in place of a user-typed character. When the `ES_PASSWORD` style is specified, the default display character is an asterisk (*).

See also: *TEdit::GetPasswordChar*, `::ES_PASSWORD`

SetReadOnly

```
inline void SetReadOnly(BOOL readOnly);
```

Sets the edit control to be read-only or read-write.

SetRect

```
inline void SetRect(const TRect& frmtRect);
```

Sets the formatting rectangle for a multiline edit control.

See also: *TEdit::GetRect*, *TEdit::SetRectNP*

SetRectNP

```
inline void SetRectNP(const TRect& frmtRect);
```

Sets the formatting rectangle for a multiline edit control. Unlike *SetRect*, *SetRectNP* does not repaint the edit control.

See also: *TEdit::GetRect*, *TEdit::SetRect*

SetSelection

```
inline BOOL SetSelection(UINT startPos, UINT endPos);
```

Forces the selection of the text between the positions specified by *startPos* and *endPos*, but not including the character at *endPos*.

SetTabStops

```
void inline SetTabStops(int numTabs, const int far* tabs);
```

Sets the tab stop positions in a multiline edit control.

SetWordBreakProc

```
inline void SetWordBreakProc(EDITWORDBREAKPROC proc);
```

In a multiline edit control, *SetWordBreakProc* indicates that an application-supplied word-break function has replaced the default word-break function. The application-supplied word-break function might break the words in the text buffer at a character other than the default blank character.

See also: *TEdit::GetWordBreakProc*

Undo

```
void Undo();
```

Undoes the last edit.

See also: *TEdit::CanUndo*, *TEdit::CMEditUndo*

UnlockBuffer

```
void UnlockBuffer(const char far* buffer, BOOL updateHandle=FALSE);
```

Unlocks a locked edit control buffer. If the contents were changed, *updateHandle* should be TRUE.

See also: *TEdit::LockBuffer*

Protected data members

Validator

```
TValidator* Validator;
```

Points to the validator object constructed in your derived class to validate input text. If no validator exists, *Validator* is zero.

Protected member functions

CanClose

```
BOOL CanClose();
```

Checks to see if all child windows can be closed before closing the current window. If any child window returns FALSE, *CanClose* returns FALSE and

terminates the process. If all child windows can be closed, *CanClose* returns TRUE.

CmEditClear

void CmEditClear();

Automatically responds to a menu selection with a menu ID of CM_EDITCLEAR by calling *Clear*.

See also: *TStatic::Clear*

CmEditCopy

void CmEditCopy();

Automatically responds to a menu selection with a menu ID of CM_EDITCOPY by calling *Copy*.

See also: *TEdit::Copy*

CmEditCut

void CmEditCut();

Automatically responds to a menu selection with a menu ID of CM_EDITCUT by calling *Cut*.

See also: *TEdit::Cut*

CmEditDelete

void CmEditDelete();

Automatically responds to a menu selection with a menu ID of CM_EDITDELETE by calling *DeleteSelection*.

See also: *TEdit::DeleteSelection*

CmEditPaste

void CmEditPaste();

Automatically responds to a menu selection with a menu ID of CM_EDITPASTE by calling *Paste*.

See also: *TEdit::Paste*

CmEditUndo

void CmEditUndo();

Automatically responds to a menu selection with a menu ID of CM_EDITUNDO by calling *Undo*.

See also: *TEdit::Undo*

ENErrSpace

void ENErrSpace();

Sounds a beep in response to an error notification message that is sent when the edit control unsuccessfully attempts to allocate more memory.

EvChar

void EvChar(UINT key, UINT repeatCount, UINT flags);

Validates the text entered into the edit control. If the input is incorrect, the original text is restored. Otherwise, the validated and modified text is placed back into the edit control, so the results of the auto-fill (if any) can be

viewed. When *IsValidInput* is called, the *SupressFill* parameter defaults to False, so that the string can be modified.

EvGetDlgCode

```
UINT EvGetDlgCode();
```

Responds to the *GetDlgCode* query according to the current state of the control. If the edit control contains valid input, then Tabs are allowed for changing focus.

EvKeyDown

```
void EvKeyDown(UINT key, UINT repeatCount, UINT flags);
```

EvKeyDown translates the virtual key code into a movement. *key* indicates the virtual key code of the pressed key, *repeatCount* holds the number of times the same key is pressed, *flags* contains one of the messages that translates to a virtual key (VK) code for the mode indicators. If the *Tab* key is sent to the Edit Control, *EvKeyDown* checks the validity before allowing the focus to change.

EvKillFocus

```
void EvKillFocus(HWND hWndGetFocus);
```

In response to a WM_KILLFOCUS message sent to a window that is losing the keyboard, *EvKillFocus* hides and then destroys the caret. *EvKillFocus* validates text whenever the focus is about to be lost and holds onto the focus if the text is not valid. Doesn't kill the focus if another application, a Cancel button, or an OK button (in which case *CanClose* is called to validate text) has the focus.

GetClassName

```
char far* GetClassName();
```

Returns the name of *TEdit*'s Windows registration class, "EDIT."

See also: *TWindow::GetClassName*

SetupWindow

```
void SetupWindow();
```

If the *textLen* data member is nonzero, *SetupWindow* limits the number of characters that can be entered into the edit control to *textLen* - 1.

See also: *TStatic::TextLen*, *TWindow::SetupWindow*

Response table entries

Response table entry	Member function
EV_COMMAND (CM_EDITCLEAR, CmEditClear)	CmEditClear
EV_COMMAND (CM_EDITCOPY, CmEditCopy)	CmEditCopy
EV_COMMAND (CM_EDITCUT, CmEditCut)	CmEditCut
EV_COMMAND (CM_EDITDELETE, CmEditDelete)	CmEditClear
EV_COMMAND (CM_EDITPASTE, CmEditPaste)	CmEditPaste

Response table entry	Member function
EV_COMMAND (CM_EDITUNDO, CmEditUndo)	CmEditUndo
EV_COMMAND_ENABLE(CM_EDITCLEAR, CmCharsEnable)	CmCharsEnable
EV_COMMAND_ENABLE(CM_EDITCOPY, CmSelectEnable)	CmSelectEnable
EV_COMMAND_ENABLE(CM_EDITCUT, CmSelectEnable)	CmSelectEnable
EV_COMMAND_ENABLE(CM_EDITDELETE, CmSelectEnable)	CmSelectEnable
EV_COMMAND_ENABLE(CM_EDITPASTE, CmPasteEnable)	CmPasteEnable
EV_COMMAND_ENABLE(CM_EDITUNDO, CmModEnable)	CmModEnable
EV_NOTIFY_AT_CHILD (EN_ERRSPACE, ENErrSpace)	ENErrSpace
EV_WM_CHAR	EvChar
EV_WM_GETDLGCODE	EvGetdlgcode
EV_WM_KEYDOWN	EvKeydown
EV_WM_KILLFOCUS	EvKillFocus
EV_WM_CHILDINVALID	EvChildInvalid

TEditFile class

editfile.h

TEditFile is a file-editing window. *TEditFile*'s data members and member functions manage the file dialog box and automatic responses for file commands such as Open, Read, Write, Save, and SaveAs. *TEditFile* is streamable.

Public data members

FileData

TOpenSaveDialog::TData FileData;

Contains information about the user's file open or save selection.

See also: *TOpenSaveDialog::TData*

FileName

char far* FileName;

Contains the name of the file being edited.

Public constructors and destructor

Constructor

```
TEditFile(TWindow* parent = 0, int Id = 0, const char far* text = 0,
          const char far* fileName = 0, TModule* module = 0);
```

Constructs a *TEditFile* window given the parent window, resource ID (*Id*), text, file name, and module ID. Sets *Filename* to *fileName*.

Destructor

```
~TEditFile();
```

Frees memory allocated to hold the name of the *TEditFile*.

Public member functions

- CanClear** virtual BOOL CanClear();
Returns TRUE if the text of the associated edit control can be cleared.
- CanClose** virtual BOOL CanClose();
Returns TRUE if the edit window can be closed.
- CmFileNew** inline void CmFileNew();
Calls *NewFile* in response to an incoming File New command with a CM_FILENEW command identifier.
See also: *TEditFile::NewFile*
- CmFileOpen** inline void CmFileOpen();
Calls *Open* in response to an incoming File Open command with a CM_FILEOPEN command identifier.
See also: *TEditFile::Open*
- CmFileSave** inline void CmFileSave();
Calls *Save* in response to an incoming File Save command with a CM_FILESAVE command identifier.
See also: *TEditFile::Save*
- CmFileSaveAs** inline void CmFileSaveAs();
Calls *SaveAs* in response to an incoming File SaveAs command with a CM_FILESAVEAS command identifier.
See also: *TEditFile::SaveAs*
- NewFile** void NewFile();
Begins the edit of a new file after calling *CanClear* to determine that it is safe to clear the text of the editor.
See also: *TEditFile::CanClear*
- Open** Open();
Opens a new file after determining that it is OK to clear the text of the *Editor*. Calls *CanClear*, and if TRUE is returned, brings up a file dialog box to retrieve the name of a new file from the user. Calls *ReplaceWith* to pass the name of the new file.

See also: *TEditFile::CanClear*, *TEditFile::ReplaceWith*

Read

```
BOOL Read(const char far* fileName=0);
```

Reads the contents of a previously specified file into the *Editor*. Returns TRUE if read operation is successful.

ReplaceWith

```
void ReplaceWith(const char far* fileName);
```

Calls *SetFileName* and *Read* to replace the file currently being edited with a file whose name is supplied.

See also: *TEditFile::SetFileName*, *TEditFile::Read*

Save

```
BOOL Save();
```

Saves changes to the contents of the *Editor* to a file. If *Editor->IsModified* returns FALSE, *Save* returns TRUE, indicating there have been no changes since the last open or save.

See also: *TEditFile::SaveAs*, *TEditFile::Write*

SaveAs

```
BOOL SaveAs();
```

Saves the contents of the *Editor* to a file whose name is retrieved from the user, through execution of a File Save dialog box. If the user selects OK, *SaveAs* calls *SetFileName* and *Write*. Returns TRUE if the file was saved.

See also: *TEditFile::SetFileName*, *TEditFile::Write*

SetFileName

```
void SetFileName (const char far* fileName);
```

Sets *FileName* and updates the caption of the window.

Write

```
BOOL Write(const char far* fileName=0);
```

Saves the contents of the *Editor* to a file whose name is specified by *FileName*. Returns TRUE if the write operation is successful.

Protected member functions

SetupWindow

```
void SetupWindow();
```

Creates the edit window's *Editor* edit control by calling *TEditFile::SetupWindow*. Sets the window's caption to *FileName*, if available; otherwise sets the name to "Untitled."

See also: *TEditFile::SetFileName*, *TEditFile::Read*

Response table entries

Response table entry	Member function
EV_COMMAND (CM_FILENEW, CmFileNew)	CmFileNew
EV_COMMAND (CM_FILEOPEN, CmFileOpen)	CmFileOpen
EV_COMMAND (CM_FILESAVE, CmFileSave)	CmFileSave
EV_COMMAND (CM_FILESAVEAS, CmFileSaveAs)	CmFileSaveAs

TEditSearch class

editsear.h

TEditSearch is an edit control that responds to Find, Replace, and FindNext menu commands. This class is streamable.

Public data members

SearchCmd

UINT SearchCmd;

Contains the search command identifier that opened the dialog box if one is open.

SearchData

TFindReplaceDialog::TData SearchData;

The *SearchData* structure defines the search text string, the replacement text string, and the size of the text buffer.

See also: *TFindReplaceDialog::TData*

SearchDialog

TFindReplaceDialog* SearchDialog;

Contains find or replace dialog-box information (such as the text to find and replace) and check box settings.

Public constructors

Constructor

```
TEditSearch(TWindow* parent = 0, int Id = 0, const char far* text = 0,
            TModule* module = 0);
```

Constructs a *TEditSearch* object given the parent window, resource ID, and character string (*text*).

Public member functions

CmEditFind

```
void CmEditFind();
```

Opens a *TFindDialog* in response to an incoming Find command with a `CM_EDITFIND` command.

CmEditFindNext

```
void CmEditFindNext();
```

Responds to an incoming FindNext command with a `CM_EDITFINDNEXT` command identifier by calling *DoSearch* to repeat the search operation.

See also: *TEditSearch::DoSearch*

CmEditReplace

```
void CmEditReplace();
```

Opens a *TReplaceDialog* in response to an incoming Replace command with a `CM_EDITREPLACE` command.

DoSearch

```
void DoSearch();
```

Performs a search or replace operation base on information in *SearchData*.

See also: *TFindReplaceDialog::TData*

EvFindMsg

```
LRESULT EvFindMsg(WPARAM, LPARAM);
```

Responds to a message sent by the modeless find or replace dialog box. Calls *DoSearch* to continue searching if text is not found or the end of the document has not been reached.

See also: *TEditSearch::DoSearch*

SetupWindow

```
void SetupWindow();
```

Posts a `CM_EDITFIND` or a `CM_EDITREPLACE` message to re-open a find or replace modeless dialog box. Calls *TEdit::SetupWindow*.

See also: *Tedit::SetupWindow*

Response table entries

Response table entry	Member function
<code>EV_COMMAND(CM_EDITFIND, CmEditFind)</code>	<code>CmEditFind</code>
<code>EV_COMMAND(CM_EDITFINDNEXT, CmEditFindNext)</code>	<code>CmEditFindNext</code>
<code>EV_COMMAND(CM_EDITREPLACE, CmEditReplace)</code>	<code>CmEditReplace</code>
<code>EV_REGISTERED(FINDMSGSTRING, EvFindMsg)</code>	<code>EvFindMsg</code>

TEditView class**editview.h**

Derived from *TView* and *TEditSearch*, *TEditView* provides a view wrapper for ObjectWindows text edit class (*TEdit*). A streamable class, *TEditView* includes several event-handling functions that handle messages between a document and its views.

Public constructors and destructor**Constructor**

```
TEditView(TDocument& doc, TWindow* parent = 0);
```

Creates a *TEditView* object associated with the specified document and parent window. Sets *Attr.AccelTable* to *IDA_EDITVIEW* to identify the edit view. Sets *TView::ViewMenu* to the new *TMenuDescr* for this view.

Destructor

```
~TEditView()
```

Destroys a *TEditView* object.

Public member functions**CanClose**

```
inline BOOL CanClose();
```

Returns nonzero if the view can be closed.

See also: *TEditFile::CanClose*

Create

```
BOOL Create()
```

Overrides *TWindow::Create* and calls *TEditSearch::Create* to create the view's window. Calls *GetDocPath* to determine if the file is new or already has data. If there is data, calls *LoadData* to add the data to the view. If the view's window can't be created, *Create* indicates the view is invalid.

GetViewName

```
inline LPCSTR GetViewName();
```

Overrides *TView::GetViewName* and returns the descriptive name of the class (*StaticName*).

See also: *TEditView::StaticName*, *TView::GetViewName*

GetWindow

```
inline TWindow* GetWindow();
```

GetWindow overrides *GetWindow* in *TView* and returns **this** as a *TWindow*.

See also: *TView::GetWindow*

PerformCreate

```
void PerformCreate(int menuOrId);
```

Allocates memory as necessary so that *TEditView* can handle files up to and including 30,000 bytes.

SetDocTitle

```
inline BOOL SetDocTitle(LPCSTR docname, int index)
```

Overrides *TView::SetDocTitle* and forwards the title to its base class, *TEditSearch*.

See also: *TWindow::SetDocTitle*, *TView::SetDocTitle*

StaticName

```
inline static LPCSTR StaticName();
```

Returns "Edit View," the descriptive name of the class for the ViewSelect menu.

Protected data member

Origin

```
long Origin;
```

Holds the file position at the beginning of the display.

Protected member functions

EvNCDestroy

```
void EvNCDestroy();
```

EvNcDestroy is used internally by *TEditView* to manage memory.

LoadData

```
BOOL LoadData();
```

LoadData reads the view from the stream and closes the file. It returns nonzero if the view was successfully loaded. If the file can't be read, posts an error and returns 0.

VnCommit

```
BOOL VnCommit(BOOL force);
```

VnCommit commits changes made in the view to the document. If *force* is nonzero, all data, even if it's unchanged, is saved to the document.

See also: *TEditView::vnRevert*, *vnxxxx* view notification constants

VnDocClosed

```
BOOL VnDocClosed(int omode);
```

VnDocClosed indicates that the document has been closed. *mode* is one of the *ofxxxx* document open constants.

See also: *ofxxxx* document open enum, *vnxxxx* view notification constants

VnIsDirty

```
inline BOOL VnIsDirty();
```

VnIsDirty returns nonzero if changes made to the data in the view have not been saved to the document; otherwise, it returns 0.

See also: *vnxxxx* view notification constants

VnIsWindow

```
inline BOOL VnIsWindow(HWND hWnd);
```

VnIsWindow returns nonzero if the window's handle passed in *hWnd* is the same as that of the view's display window.

VnRevert

```
BOOL VnRevert(BOOL clear);
```

VnRevert is nonzero if changes made to the view should be erased, and the data from the document should be restored to the view. If *clear* is nonzero, the data is cleared instead of restored to the view.

See also: *TEditView::Commit*

Response table entries

Response table entry	Member function
EV_VN_COMMIT	VnCommit
EV_VN_DOCCLOSED	VnDocClosed
EV_VN_ISDIRTY	VnIsDirty
EV_VN_ISWINDOW	VnIsWindow
EV_WM_NCDESTROY	EvNcDestroy
EV_VN_REVERT	VnRevert

TEventHandler class

eventhan.h

TEventHandler is a base class from which you can derive classes that handle messages. See Chapter 2 for a list of the Windows messages that *ObjectWindows* handles.

Public member functions

Dispatch

```
virtual LRESULT Dispatch(TEventInfo&, WPARAM, LPARAM = 0);
```

Takes the message data from *Msg* and dispatches it to the correct event handler.

Find

```
virtual BOOL Find(TEventInfo&, TEqualOperator = 0);
```

Searches the list of response table entries looking for a match. Because *TEventHandler* doesn't have any entries, *TEventHandler's* implementation of this routine returns FALSE.

Protected member functions

SearchEntries

```
BOOL SearchEntries(TGenericTableEntry __RTFAR* entries, TEventInfo&,
                  TEqualOperator);
```

Searches the entries in the response table for an entry that matches *TEventInfo* or, if so designated, an entry that *TEqualOperator* specifies is a match.

TEventHandler::TEqualOperator type

eventhan.h

A nested class, *TEqualOperator* is used to perform special kinds of searches and to facilitate finding response table entries. *TEqualOperator* compares a particular message event (*TEventInfo&*) with a response table entry (*TGenericTableEntry*) to determine if they match.

```
typedef BOOL(*TEqualOperator)(TGenericTableEntry __RTFAR&, TEventInfo&);
```

See also: *TResponseTableEntry*

TEventHandler::TEventInfo class

eventhan.h

A nested class, *TEventInfo* provides specific information about the type of message sent, the class that contains the function to be handled, the corresponding response table entry, and the dispatch function that processes the message.

Public data members

Dispatch

```
LRESULT Dispatch(GENERIC &object, TGenericTableEntry &entry,
                 WPARAM wParam, LPARAM lParam);
```

Generic dispatch function to crack and dispatch Windows messages. *GENERIC &object* is the pointer to the object (for example, *TEdit*); *TGenericTableEntry &entry* is the response table entry (for example, *EvActivate*); *WPARAM* and *LPARAM* are the message parameters the dispatcher cracks.

Find

```
virtual TGenericTableEntry *Find(GENERIC *&object, UINT msg, UINT Id,
                                TEqualOperator = 0);
```

Find locates and returns an entry that a *TEventHandler* class can respond to.

Msg

```
const UINT Msg;
```

Contains the type of message sent. These can be command messages, child id messages, notify-based messages such as LBN_SELCHANGE, or windows messages such as LBUTTONDOWN.

Id const UINT Id;

Contains the menu or accelerator resource ID (CM_XXXX) for the message response member function.

Object GENERIC *Object;

Points to the object that contains the function to be handled.

Entry TGenericTableEntry __RTFAR *Entry;

Points to the response table entry, for example *EvActivate*.

Public constructors

Constructor TEventInfo(UINT msg, UINT id = 0) : Msg(msg), Id(id);

Constructs a *TEventInfo* object with the specified ID and message type.

TEventStatus enum

window.h

Event status constants indicate the status of a mix-in window event implementation. For example, depending on the implementation of a keyboard event, *DoKeyDown* and *DoSetFocus* return one of *TEventStatus* constants.

Table 1.27
Event status
constants

Constant	Meaning
esPartial	Additional handlers can be invoked
esComplete	No additional handlers are needed

See also: *TKeyboardModeTracker::DoKeyDown*,
TKeyboardModeTracker::DoSetFocus

TEventHandler::TEqualOperator type

eventhan.h

A nested class, *TEqualOperator* is used to perform special kinds of searches and to facilitate finding response table entries. *TEqualOperator* compares a particular message event (*TEventInfo&*) with a response table entry (*TGenericTableEntry*) to determine if they match.


```
typedef BOOL(*TEqualOperator)(TGenericTableEntry __RTFAR&, TEventInfo&);
```

See also: *TResponseTableEntry*

TFileDocument class

filedoc.h

Derived from *TDocument*, *TFileDocument* opens and closes views and provides stream support for views. Streams are created on top of DOS files using Windows file services. *TFileDocument* has member functions that continue to process *FileNew* and *FileOpen* messages after a view is constructed. You can add support for specialized file types by deriving classes from *TFileDocument*. *TFileDocument* makes this process easy by hiding the actual process of storing file types.

Public constructors and destructor

Constructor

```
TFileDocument(TDocument* parent = 0);
```

Constructs a *TFileDocument* object with the optional parent document.

Destructor

```
~TFileDocument();
```

Destroys a *TFileDocument* object.

Public member functions

Close

```
inline BOOL Close();
```

Closes the document but does not delete or detach any associated views. Before closing the document, *Close* calls *TDocument's Close* to make sure all child documents are closed. If any children are open, *Close* returns 0 and doesn't close the document. If all children are closed, checks to see if any associated streams are open, and if so, returns 0 and doesn't close the document. If there are no open streams, closes the file.

See also: *TDocument::Close*

Commit

```
BOOL Commit(BOOL clear = FALSE);
```

Calls *TDocument::Commit* and clears *TDocument's DirtyFlag* data member, thus indicating that there are no unsaved changes made to the document.

See also: *TDocument::Commit*

FindProperty

```
int FindProperty(const char far* name);
```

FindProperty gets the property index, given the property name (*name*). Returns 0 if the name isn't found.

See also: *pfxxxx* property attribute constants

GetProperty

```
GetProperty(int index, void far* dest, int textlen=0);
```

Overrides *TDocument::GetProperty* and gets the property ID for the current file document.

See also: *pfxxxx* property attribute constants

InStream

```
TInStream* InStream(int mode, LPCSTR strmId = 0);
```

Overrides *Tdocument::InStream* and provides generic input for the particular storage medium. *InStream* returns a pointer to a *TInStream*. *mode* is a combination of the ios bits defined in *iostream.h*. *strmId* is not used for file documents. The view reads data from the document as a stream or through stream functions.

See also: *TFileDocument::OutStream*

IsOpen

```
inline BOOL IsOpen();
```

Is nonzero if the document or any streams are open.

Open

```
inline BOOL Open(HFILE fhdl);
```

Opens a file document using an existing file handle. Sets *TDocument::OpenMode* to *PREV_OPEN* and read/write. Sets the document path to 0. Sets *FHd* to *fhdl*. Always returns nonzero.

Open

```
inline BOOL Open(int mode, LPCSTR path=0);
```

Overrides *TDocument::Open* and opens the file using the specified *path*. If the file is already open, returns 0. Calls *TDocument::SetDocPath* to set the directory path. If *omode* isn't 0, sets *TDocument::OpenMode* to *omode*. If the file can't be opened, returns 0.

See also: *TDocument::DocPath*, *TDocument::Open*, *ofxxxx* document open enum

OutStream

```
inline TOutStream* OutStream (int mode, LPCSTR strmId = 0);
```

Overrides *TDocument::OutStream* and provides generic input for the particular storage medium. *OutStream* returns a pointer to a *TOutStream*. *mode* is a combination of the ios bits defined in *iostream.h*. *strmId* is not used for file documents. Instead, the view reads data from the document through stream functions.

See also: *TFileDocument::InStream*

PropertyFlags

```
int PropertyFlags(int index);
```

Returns the property attribute constants (*pfGetText*, *pfHidden*, and so on).

See also: *pfxxxx property attribute constants*

PropertyName

```
const char* PropertyName(int index);
```

Returns the text name of the property given the index value.

See also: *pfxxxx property attribute constants*

Revert

```
BOOL Revert(BOOL clear = FALSE);
```

Calls *TDocument::Revert* to notify the views to refresh their data. If *clear* is *FALSE*, the data is restored instead of cleared.

See also: *TFileDoc::Commit*

SetProperty

```
BOOL SetProperty(int index, const void far* src);
```

Sets the property data, which must be in the native data type (either string or binary).

See also: *pfxxxx property attribute constants*

Protected data members

FHdl

```
HFILE FHdl;
```

Holds the file handle to an open file document.

Protected member functions

CloseThisFile

```
void CloseThisFile(HFILE fhdl, into omode);
```

If the associated file was opened by *TFileDocument*, *CloseThisFile* closes the file handle. Calls *TDocument::NotifyView* to notify all views that the file document has closed.

See also: *ofxxxx document open enum*

OpenThisFile

```
HFILE OpenThisFile(int omode, LPCSTR name, streampos* pseekpos);
```

Opens the file document after checking the file sharing mode (*omode*). If a file mode is not specified as read, write, or read and write, returns 0.

See also: *ofxxxx document open enum*, *shxxxx document sharing modes*

TFileOpenDialog class**opensave.h**

TFileOpenDialog is a modal dialog box that lets you specify the name of a file to open.

Public constructors**Constructor**

```
TFileOpenDialog(TWindow* parent, TData& data, TResID templateID = 0,
               const char far* title = 0, TModule* module = 0);
```

Constructs and initializes the *TFileOpen* object based on information in the *TOpenSaveDialog::TData* data structure. The Windows API OPENFILENAME structure is also initialized.

See also: *TOpenSaveDialog::TData*

Public member functions**DoExecute**

```
int DoExecute();
```

DoExecute calls the Windows API function *GetOpenFileName* and passes it the Windows API OPENFILENAME structure, which tells it how to create the *TFileOpenDialog* object.

See also: *TDialog::DoExecute*

TFileSaveDialog class**opensave.h**

TFileSaveDialog is a modal dialog that lets you enter the name of a file to save.

Public constructors**Constructor**

```
TFileSaveDialog(TWindow* parent, TData& data, TResID templateID = 0,
               const char far* title = 0, TModule* module = 0);
```

Constructs and initializes the *TFileOpen* object based on information in the *TOpenSaveDialog::TData* data structure.

Public member functions

DoExecute

```
int DoExecute();
```

DoExecute calls the Windows API function *GetOpenFileName* and passes it the OPENFILENAME structure, which tells *GetOpenFileName* how to create the *TFileSaveDialog* object.

See also: *TDialog::DoExecute*, *TOpenSaveDialog*

TFilterValidator class

validate.h

A streamable class, *TFilterValidator* checks an input field as the user types into it. The validator holds a set of allowed characters. When the user enters a character, the filter validator indicates whether the character is valid or invalid. See *TValidator* for an example of input validation.

Public constructors

Constructor

```
TFilterValidator(const TCharSet& validChars);
```

Constructs a filter validator object by first calling the constructor inherited from *TValidator*, then setting *ValidChars* to *validChars*.

Public member functions

Error

```
void Error();
```

Error overrides *TValidator*'s virtual function and displays a message box indicating that the text string contains an invalid character.

See also: *TValidator::Error*

IsValid

```
BOOL IsValid(const char far* str);
```

IsValid overrides *TValidator*'s virtuals and returns TRUE if all characters in *str* are in the set of allowed characters, *ValidChar*; otherwise, it returns FALSE.

IsValidInput

```
BOOL IsValidInput(char far* str, BOOL suppressFill);
```

IsValidInput overrides *TValidator*'s virtual function and checks each character in the string *str* to ensure it is in the set of allowed characters, *ValidChar*. *IsValidInput* returns TRUE if all characters in *str* are valid; otherwise, it returns FALSE.

See also: *TValidator::IsValidInput*

Protected data members

ValidChars

TCharSet ValidChars;

Contains the set of all characters the user can type. For example, to allow only numeric digits, set *ValidChars* to "0-9". *ValidChars* is set by the *validChars* parameter passed to the constructor.

TFindDialog class

findrepl.h

TFindDialog objects represents modeless dialog box interface elements that let you specify text to find. *TFindDialog* communicates with the owner window using a registered message.

Public constructors

Constructor

```
TFindDialog(TWindow* parent, TData& data, TResId templateId = 0,
            const char far* title = 0, TModule* module = 0);
```

Constructs a *TFindDialog* object with the given parent window, resource ID, and caption. Sets the attributes of the dialog box based on information in the *TFindReplaceDialog::TData* structure.

See also: *TFindReplaceDialog::TData*

Protected member functions

DoCreate

```
HWND DoCreate();
```

Creates the modeless interface element of a find dialog box.

TFindReplaceDialog class

findrepl.h

TFindReplaceDialog is an abstract base class for a modeless dialog box that lets you search for and replace text. *TFindReplaceDialog* communicates with the owner window using a registered message.

Public constructors

Constructor

```
TFindReplaceDialog(TWindow* parent, TData& data, TResId templateId = 0,
                  const char far* title = 0, TModule* module = 0);
```

Constructs a *TFindReplaceDialog* object with a parent window, resource ID, and caption. Sets the attributes of the dialog box with the specified data.

See also: *TFindReplaceDialog::TData*

Public member functions

CmCancel

```
inline void CmCancel();
```

Responds to a click of the Cancel button.

CmFindNext

```
inline void CmFindNext();
```

Responds to a click of the Find Next button.

CmReplace

```
inline void CmReplace();
```

Responds to a click of the Replace button.

CmReplaceAll

```
inline void CmReplaceAll();
```

Responds to a click of the Replace All button.

EvNCDestroy

```
inline void EvNCDestroy();
```

Calls *TWindow::EvNCDestroy*, which responds to an incoming *EV_WM_NCDESTROY* message.

See also: *TWindow::EvNCDestroy*

Protected data members

Data

```
TData& Data;
```

Data is a reference to the *TData* object passed in the constructor.

See also: *TFindReplace::TData*

fr

```
FINDREPLACE fr;
```

Contains find-and-replace attributes such as the text string to be searched for and replaced as well as the length of the string that ObjectWindows passes to the Windows API *FindText* function. Returns an error message if text is not found.

Protected member functions

DoCreate

HWND DoCreate()=0;

DoCreate is a virtual function that is overridden in derived classes to create a modeless find or replace dialog box.

DialogFunction

BOOL DialogFunction(UINT message, WPARAM, LPARAM);

Returns TRUE if a message is handled.

See also: *TDialog::DialogFunction*

Init

void Init(TResId templateId);

Used by constructors in derived classes, *Init* initializes a *TFindReplaceDialog* object with the current resource ID and other members.

Response table entries

Response table entry	Member function
EV_WM_NCDESTROY	EvNCDestroy

TFindReplaceDialog::TData struct

findrepl.h

The *TFindReplaceDialog::TData* structure defines information necessary to initialize a *TFindReplace* dialog box.

Public member functions

BuffSize

int BuffSize;

BuffSize contains the size of the text buffer.

Error

DWORD Error;

Error contains one or more of the following *CommDlgExtendedError* codes:

Constant	Meaning
CDERR_LOCKRESOURCEFAILURE	Failed to lock a specified resource.
CDERR_LOADRESFAILURE	Failed to load a specified resource.

Constant	Meaning
CDERR_LOADSTRFAILURE	Failed to load a specified string.
CDERR_REGISTERMSGFAIL	The Windows API <i>RegisterWindowMessage</i> function returned an error when it was called.

FindWhat

char* FindWhat;

FindWhat contains the search string.

Flags

DWORD Flags;

Flags indicates the state of the control buttons and the action that occurred in the dialog box, and can be a combination of the following Window API constants:

Constant	Meaning
FR_DOWN	The Down button in the Direction group of the Find dialog box is selected.
FR_HIDEMATCHCASE	The Match Case check box is hidden.
FR_HIDEWHOLEWORD	The Whole Word check box is hidden.
FR_HIDEUPDOWN	The Up and Down buttons are hidden.
FR_MATCHCASE	The Match Case check box is checked.
FR_NOMATCHCASE	The Match Case check box is disabled. This occurs when the dialog box is first initialized.
FR_NOUPDOWN	The Up and Down buttons are disabled. This occurs when the dialog box is first initialized.
FR_NOWHOLEWORD	The Whole Word check box is disabled. This occurs when the dialog box is first initialized.
FR_REPLACE	The Replace button was pressed in the Replace dialog box.
FR_REPLACEALL	The Replace All button was pressed in the Replace dialog box.
FR_WHOLEWORD	The Whole Word check box is checked.

ReplaceWith

char* ReplaceWith;

ReplaceWith contains the replacement string.

See also: *TEditSearch::SearchData*, *TFindReplace::Data*

TFloatingFrame class**floatfra.h**

Derived from *TFrameWindow* and *TTinyCaption*, *TFloatingFrame* implements a floating frame that can be positioned anywhere in the parent window. Except for the addition of a tiny caption bar, the default behavior of *TFrameWindow* and *TFloatingFrame* is the same. Therefore, an application that uses *TFrameWindow* can easily gain the functionality of *TFloatingFrame* by just changing the name of the class to *TFloatingFrame*.

If there is a client window, the floating frame shrinks to fit the client window, leaving room for margins on the top, bottom, left, and right of the frame. Because the floating frame expects the client window to paint its own background, it does nothing in response to a `WM_ERASEBKGND` message. However, if there is no client window, the floating frame erases the client area background using `COLOR_BTNFACE`.

See `PAINT.CPP`, the sample program on your distribution disk, for an example of a floating frame.

Public constructors

Constructor

```
TFloatingFrame(TWindow *owner, char *title = 0, TWindow* clientWnd = 0,
               BOOL shrinkToClient = FALSE, int CaptionHeight, BOOL
               enablePalette, Module* module = 0);
```

Constructs a *TFloatingFrame* object attached to the specified parent window. By default, the floating frame window doesn't shrink to fit the client window, and the floating palette style isn't enabled.

Set *enablePalette* to `TRUE` if you want to enable a floating palette style for the window. The floating palette is a popup window with a tiny caption, a standard window border, and a close box instead of a system menu box. There are no maximize or minimize buttons. A one pixel border is added around the client area in case a toolbox is implemented. This style must be turned on before the window is created. After the window is created, its style can't be changed.

See also: *TFrameWindow::TFrameWindow*, *TTinyCaption::TTinyCaption*

Public member functions

SetMargins

```
void SetMargins(const TSize& margin);
```

Sets the margins of the floating palette window to the size specified in *margin* and sets the height of the tiny caption bar.

See also: *TTinyCaption::EnableTinyCaption*

Response table entries

Response table entry	Member function
EV_WM_SYSCOMMAND	EvSysCommand
EV_WM_NCCALCSIZE	EvNCCalcSize
EV_WM_NCPAINT	EvNCPaint

TFont class

gdiobjec.h

TFont derived from *TGdiObject* provides constructors for creating font objects from explicit information or indirectly.

Public data members

enum TStockId

```
enum TStockId{AnsiFixed, AnsiVar, DeviceDefault, OemFixed, System,
              SystemFixed};
```

Enumerates the stock fonts.

See also: *TGdiObject::Stocks[]*

Protected data members

Stocks[]

```
static TFont Stocks[];
```

Note: This array no longer exists. Use *TDC::SelectStockObject* instead.

The single static array of Windows stock fonts serving all *TFont* objects. The stock fonts are *ANSI_FIXED_FONT*, *ANSI_VAR_POINT*, *DEVICE_DEFAULT_FONT*, *OEM_FIXED_FONT*, *SYSTEM_FONT*, and *SYSTEM_FIXED_FONT*.

See also: *enum TStockId*

Public constructors

Constructor

```
TFont(HFONT handle, TAutoDelete autoDelete = NoAutoDelete);
```

Creates a *TFont* object and sets the *Handle* data member to the given borrowed *handle*. The *ShouldDelete* data member defaults to *FALSE*, ensuring that the borrowed handle will not be deleted when the C++ object is destroyed.

See also: *TGdiObject::Handle*, *TGdiObject::ShouldDelete*

Constructor

```
TFont(const char far* facename = 0, int height = 0, int width = 0,
      int escapement = 0, int orientation = 0, int weight = FW_NORMAL,
      BYTE pitchAndFamily = DEFAULT_PITCH|FF_DONTCARE, BYTE italic
      = FALSE, BYTE underline = FALSE, BYTE strikeout = FALSE,
      BYTE charSet = 1, BYTE outputPrecision = OUT_DEFAULT_PRECIS,
      BYTE clipPrecision = CLIP_DEFAULT_PRECIS, BYTE quality
      = DEFAULT_QUALITY);
```

Creates a *TFont* object with the given values. Sets *Handle* via a Win API *CreateFont* call with the given default values.

See also: `::CreateFont`

Constructor

```
TFont(const LOGFONT far* logFont);
```

Creates a *TFont* object from the given *logFont*. Sets *Handle* via a WinAPI *CreateFontIndirect(logFont)* call.

See also: `::CreateFontIndirect`

Public member functions

GetObject

```
inline BOOL GetObject(LOGFONT far& logFont) const;
```

Retrieves information about this pen object and places it in the given *LOGFONT* structure. Returns TRUE if successful and FALSE if unsuccessful.

See also: `TGdiObject::GetObject`, struct *LOGFONT*

GetStock

```
static TFont& GetStock(TStockId id);
```

Provides access to stock Windows font objects. Returns *TFont::Stocks[id]*.

See also: enum *TStockId*

operator HFONT()

```
inline operator HFONT() const;
```

Typecasting operator that converts this font's *Handle* to type *HFONT* (the Windows data type representing the handle to a physical font).

TFrameWindow class

framewin.h

Derived from *TWindow*, *TFrameWindow* controls such window-specific behavior as keyboard navigation and command processing for client windows. For example, when a window is reactivated, *TFrameWindow* is

responsible for restoring a window's input focus and for adding menu bar and icon support. *TFrameWindow* is a streamable class.

See *TFloatingFrame* for a description of a floating frame with the same default functionality as a frame window.

Public data member

KeyboardHandling

BOOL KeyboardHandling;

Indicates if keyboard navigation is required.

Public constructors and destructor

Constructor

```
TFrameWindow(TWindow* parent, const char far *title = 0,
             TWindow *clientWnd = 0, BOOL shrinkToClient = FALSE,
             TModule* module = 0);
```

Constructs a window object with the parent window supplied in *parent*. Sets the position and extent fields of *Attr* structure to defaults appropriate for overlapped and pop-up windows.

See also: *TWindow::TWindow*, *TFloatingFrame::TFloatingFrame*

Constructor

```
TFrameWindow(HWND hWnd, TModule* module = 0);
```

Constructor for a *TFrameWindow* that is being used as an alias for a non-ObjectWindows window.

Destructor

```
~TFrameWindow;
```

Deletes any associated menu descriptor.

Public member functions

AssignMenu

```
virtual BOOL AssignMenu(TResId menuResId);
```

Sets *Attr.Menu* to the supplied *menuResId* and frees any previous strings pointed to by *Attr.Menu*. If *HWindow* is nonzero, loads and sets the menu of the window, destroying any previously existing menu.

See also: *TMDIFrame::AssignMenu*, *TFrameWindow::SetMenu*

EnableKBHandler

```
void EnableKBHandler();
```

Sets a flag indicating that the receiver has requested keyboard navigation (translation of keyboard input into control selections). By default, the

keyboard interface, which lets users use the tab and arrow keys to move between the controls, is disabled for windows and dialog boxes.

- GetClientWindow** inline virtual TWindow *GetClientWindow();
Returns a pointer to the client window.
See also: *TMDIClient::GetClientWindow*
- GetMenuDescr** inline const TMenuDescr* GetMenuDescr();
Returns a pointer to the menu descriptor.
See also: *TFrameWindow::SetMenuDescr*
- HoldFocusHwnd** virtual BOOL HoldFocusHwnd(HWND hWndLose, HWND hWndGain);
Responds to a request by a child window to hold its HWND when it is losing focus. Stores the child's HWND in *HwndRestoreFocus*.
See also: *TWindow::HoldFocusHwnd*
- IdleAction** void IdleAction();
TApplication calls the main window's *IdleAction* when no messages are waiting to be processed. *TFrameWindow* uses this idle time to perform command enabling for the menu bar. It also forwards *IdleAction* to each of its children. *IdleAction* can be overridden to do background processing.
See also: *TApplication::IdleAction*
- MergeMenu** BOOL MergeMenu(const TMenuDescr& childMenuDescr);
Merges the given menu descriptor with this frame's own menu descriptor and displays the resulting menu in this frame. See *TMenuDescr* for a description of menu bar types that can be merged.
See also: *TMenuDescr* class
- PreProcessMsg** BOOL PreProcessMsg(MSG& msg);
Performs preprocessing of window messages. If the child window has requested keyboard navigation, *PreProcessMsg* handles any accelerator key messages and then calls the Windows API function *::IsDialogMessage* to process any other keyboard messages.
See also: *TWindow::PreProcessMsg*
- RestoreMenu** BOOL RestoreMenu();
Restores the default menu of the frame window.
- SetClientWindow** virtual TWindow *SetClientWindow(TWindow* clientWindow);
Sets the client window to the specified window.

SetIcon `BOOL SetIcon(TInstance iconInst, TResId iconResId);`

Sets the icon to the specified resource ID.

See also: *TFrameWindow::EvQueryDragIcon*

SetMenuDescr `void SetMenuDescr(const TMenuDescr& menuDescr);`

Sets the menu descriptor to the new menu descriptor.

See also: *TFrameWindow::GetMenuDescr*

Protected data members

ClientWnd `TWindow *ClientWnd;`

ClientWnd points to the frame's client window.

DocTitleIndex `int DocTitleIndex;`

Holds the index number for the document title.

HWndRestoreFocus `HWND HWndRestoreFocus;`

Stores the handle of the child window whose focus gets restored.

See also: *TFrameWindow::HoldFocusHwnd*

Protected member functions

EvCommand `LRESULT EvCommand(UINT id, HWND hWndCtl, UINT notifyCode);`

Provides extra processing for commands and lets the focus window and its parent windows handle the command first.

EvCommandEnable `void EvCommandEnable(TCommandEnabler& ce);`

Handles checking and unchecking of the frame window's menu items.

See also: *TMenuItemEnabler::TCommandEnabler*

EvEraseBkgnd `BOOL EvEraseBkgnd(HDC);`

EvEraseBkgnd erases the background of the window specified in *HDC*. It returns TRUE if the background is erased; otherwise, it returns FALSE.

EvInitMenuPopup `HANDLE EvInitMenuPopup(HMENU hPopupMenu, UINT index, BOOL sysMenu);`

Sent before a pop-up menu is displayed, *EvInitMenuPopup* lets an application change the items on the menu before the menu is displayed. *EvInitMenuPopup* controls whether the items on the pop-up menu are

enabled or disabled, checked or unchecked, or strings. *HMENU* indicates the menu handle. *index* is the index of the pop-up menu. *sysMenu* indicates if the pop-up menu is the system menu.

See also: `::WM_INITMENUPOPUP`

EvPaint

```
void EvPaint();
```

Responds to a `WM_PAINT` message in order to paint the iconic window's icon or to allow client windows a chance to paint the icon.

See also: `TWindow::EvPaint`, `TWindow::Paint`, `TScroller::BeginView`, `TScroller::EndView`

EvParentNotify

```
void EvParentNotify(UINT event, UINT childHandleOrX, UINT childIDOrY);
```

Responds to a message to notify the parent window that a given event has occurred. If the client window is destroyed, closes the parent window. If *shrinkToClient* is set and the child window has changed size, the frame is adjusted.

EvQueryDragIcon

```
HANDLE EvQueryDragIcon();
```

Responds to a Windows API `WM_QUERYDRAGICON` message sent to a minimized (iconic) window that is going to be dragged. Instead of the default Windows icon, *EvQueryDragIcon* uses the icon that was set using *SetIcon*.

See also: `TFrameWindow::SetIcon`, `::WM_QUERYDRAGICON`

EvSetFocus

```
void EvSetFocus(HWND hWndLostFocus);
```

Restores the focus to the active window. If the object is an alias, *EvSetFocus* calls `TWindow::Activate`. *hWndLostFocus* contains the handle to the window that lost the focus.

See also: `TWindow::EvActivate`, `::WM_SETFOCUS`

EvSize

```
void EvSize(UINT sizeType, TSize &size);
```

Resizes the client window's size so that it is equivalent to the client rectangle's size. Calls `TWindow::EvSize` in response to an incoming Windows API `WM_SIZE` message.

See also: `TWindow::EvSize`

SetupWindow

```
void SetupWindow();
```

Calls `Twindow::SetupWindow` to create windows in a child list. *SetupWindow* performs the initial adjustment of the client window if one exists, assigns the frame's menu based on the menu descriptor, and initializes *HwndRestoreFocus*.

See also: *TWindow::SetUpWindow*

Response table entries

Response table entry	Member function
EV_WM_ERASEBKGND	EvEraseBkgnd
EV_WM_INITMENUPOPUP	EvWmInitMenuPopup
EV_WM_PAINT	EvPaint
EV_WM_PARENTNOTIFY	EvParentNotify
EV_WM_QUERYDRAGICON	EvQueryDragIcon
EV_WM_SETFOCUS	EvWmSetFocus
EV_WM_SIZE	EvWmSize

TGadget class

gadget.h

TGadget is the base class for the derived classes—*TBitmapGadget*, *TButtonGadget*, *TControlGadget*, *TTextGadget*, and *TSeparatorGadget*. *TGadget* interface objects belong to a gadget window, have borders and margins, and have their own coordinate system. The margins are the same as those for *TGadgetWindow* and borders are always measured in border units.

To set the attributes for the gadget, you can either choose a border style (which automatically sets the individual border edges) or set the borders and then override the member function *PaintBorder* to create a custom look for your gadget. If you change the borders, margins, or border style, the gadget window's *GadgetChangedSize* member function is invoked.

Although, by default, gadgets shrink-wrap to fit around their contents, you can control this attribute by setting your own values for *ShrinkWrapWidth* and *ShrinkWrapHeight*.

Public data members

Clip

BOOL Clip;

If *Clip* is FALSE, clipping borders have not been established. If *Clip* is TRUE, the drawing for each gadget is restrained by the gadget's border.

TBorders struct

TBorders structure holds the values for the left, right, top, and bottom measurements of the gadget.

```
struct TBorders
    unsigned Left;
    unsigned Right;
    unsigned Top;
    unsigned Bottom;
```

TBorderStyle enum

```
enum TBorderStyle;
```

Enumerates border styles as either none, plain, raised, recessed, or embossed. For an example of border styles, see the sample `ObjectWindows` program, `MDIFILE.CPP`, on your distribution disk.

TMargins struct

Used by the `TGadgetWindow` and `TGadget` classes, `TMargins` contains the measurements of the margins for the gadget. The constructor initializes `Units` to `LayoutUnits` and sets `Left`, `Right`, `Top`, and `Bottom` equal to 0.

```
struct TMargins {
    enum TUnits {Pixels, LayoutUnits, BorderUnits};
    TUnits Units;
    int Left;
    int Right;
    int Top;
    int Bottom;
```

See also: `TGadgetWindow::SetMargins`

WideAsPossible

```
BOOL WideAsPossible;
```

Initially set to `FALSE`, `WideAsPossible` indicates whether the gadget width will be adjusted by the gadget window to be as wide as possible in the remaining space.

See also: `TGadgetWindow::WideAsPossible`

Public constructors and destructor

Constructor

```
TGadget(int id = 0, TBorderStyle = None);
```

Constructs a `TGadget` object with the specified ID and border style.

Destructor

```
virtual ~TGadget();
```

Destroys a `TGadget` interface object and removes it from its associated window.

Public member functions

- CommandEnable** virtual void CommandEnable();
- CommandEnable* is provided so that the gadget can perform command enabling (so it can handle an incoming message, if it's appropriate to do so).
- GetBorders** inline TBorders &GetBorders();
- Gets the gadget's borders measured in border units that are based on SM_CXBORDER and SM_CYBORDER.
- See also: *TGadget::SetBorders*
- GetBorderStyle** inline TBorderStyle GetBorderStyle();
- Gets the style for the gadget's borders.
- See also: *TGadget::SetBorderStyle*
- GetBounds** inline TRect &GetBounds();
- Returns the boundary rectangle for the gadget.
- See also: *TButtonGadget::SetNotchCorners*
- GetDesiredSize** virtual void GetDesiredSize(TSize& size);
- GetDesiredSize* determines how big the gadget can be. The gadget window sends this message to query the gadget's size. If shrink-wrapping is requested, *GetDesiredSize* returns the size needed to accommodate the borders and margins. If shrink-wrapping is not requested, it returns the gadget's current width and height. *TGadgetWindow* needs this information to determine how big the gadget needs to be, but it can adjust these dimensions if necessary. If *WideAsPossible* is TRUE, then the width parameter (*size.cx*) is ignored.
- GetEnabled** inline BOOL GetEnabled();
- Determines whether keyboard and mouse input have been enabled for the specified gadget. If the gadget is enabled, *GetEnabled* returns TRUE; otherwise, it returns FALSE. By default, keyboard and mouse input are enabled.
- See also: *TGadget::SetEnabled*
- GetId** inline int GetId();
- Gets the ID for the gadget.
- GetMargins** inline TMargins &GetMargins();
- Gets the margin dimensions.

- GetOuterSizes** `void GetOuterSizes(int& left, int& right, int& top, int& bottom);`
Returns the amount of space (in pixels) taken up by the borders and margins.
- NextGadget** `inline TGadget *NextGadget();`
Returns the next gadget in the list of gadgets.
- SetBorders** `void SetBorders(TBorders& borders);`
Sets the borders for the gadget. If the borders are changed, *SetBorders* calls *TGadgetWindow::GadgetChangedSize* to notify the gadget window of the change.
See also: *TGadget::GetBorders*, *TGadgetWindow::GadgetChangedSize*
- SetBorderStyle** `void SetBorderStyle(TBorderStyle);`
Sets the border style for the gadget.
See also: *TGadget::GetBorderStyle*
- SetBounds** `virtual void SetBounds(TRect& rect);`
SetBounds informs the gadget of a change in its bounding rectangle. Although the default behavior updates only the instance variable *Bounds*, you can override this method to also update the internal state of the gadget.
- SetEnabled** `virtual void SetEnabled(BOOL);`
Enables or disables keyboard and mouse input for the gadget. By default, the gadget is disabled when it is created and must be enabled before it can be activated.
See also: *TGadget::GetEnabled*
- SetMargins** `void SetMargins(TMargins& margins);`
Sets the margins of the gadget. If the margins are changed, *SetMargins* calls *TGadgetWindow::GadgetChangedSize* to notify the gadget window.
See also: *TGadget::GetMargins*
- SetShrinkWrap** `void SetShrinkWrap(BOOL shrinkWrapWidth, BOOL shrinkWrapHeight);`
Sets the *ShrinkWrapWidth* and *ShrinkWrapHeight* data members. Your derived class can call *TGadgetWindow's GadgetChangedSize* member function if you want to change the size of the gadget.
See also: *TGadgetWindow::GadgetChangedSize*
- SetSize** `void SetSize(TSize& size);`

SetSize alters the size of the gadget and then calls *TGadgetWindow::GadgetChangedSize* for the size change to take effect.

This function is needed only if you have turned off shrink-wrapping in one or both dimensions; otherwise, use the *GetDesiredSize* member function to return the shrink-wrapped size.

SysColorChange

virtual void SysColorChange();

SysColorChange is called when the system colors have been changed so that gadgets can rebuild and repaint, if necessary.

Protected data members

Bounds

TRect Bounds;

Contains the bounding rectangle for the gadget in gadget window coordinates.

See also: *TGadget::GetInnerRect*

BorderStyle

TBorderStyle BorderStyle;

Contains the border style for the gadget.

Borders

TBorders Borders;

Contains the border measurements for the gadget.

See also: *TGadget::GetInnerRect*

Id

int Id;

Contains the gadget's ID.

Margins

TMargins Margins;

Contains the margin measurements of the rectangle.

See also: *TGadget::GetInnerRect*

ShrinkWrapHeight

BOOL ShrinkWrapHeight;

Indicates if the gadget is to be shrink-wrapped to fit around its contents.

ShrinkWrapWidth

BOOL ShrinkWrapWidth;

Indicates if the gadget is to be shrink-wrapped to fit around its contents.

TrackMouse

BOOL TrackMouse;

Initialized to FALSE. When *TrackMouse* is TRUE, the gadget captures and releases the mouse on *LButtonDown* and *LButtonUp* by calling *TGadgetWindow's GadgetSetCapture* and *GadgetReleaseCapture*.

See also: *TGadget::LButtonDown*, *TGadget::LButtonUp*

Window

*TGadgetWindow *Window*;

References the owning or parent window for the gadget.

Protected member functions

GetInnerRect

void GetInnerRect (TRect& rect);

Computes the area of the gadget's rectangle excluding the borders and margins.

Inserted

virtual void Inserted();

Called after a gadget is inserted into a window.

Invalidate

void Invalidate(BOOL erase = TRUE);

Used to invalidate the active (usually nonborder) portion of the gadget, *Invalidate* calls *InvalidateRect* and passes the boundary width and height of the area to erase.

InvalidateRect

void InvalidateRect(const TRect& rect, BOOL erase = TRUE);

Invalidates the gadget-relative rectangle in the parent window.

LButtonDown

virtual void LButtonDown(UINT modKeys, TPoint& point);

Captures the mouse if *TrackMouse* is set. *point* is located in the gadget's coordinate system.

See also: *TGadget::TrackMouse*

LButtonUp

virtual void LButtonUp(UINT modKeys, TPoint& point);

Releases the mouse capture if *TrackMouse* is set. *point* is located in the gadget's coordinate system.

See also: *TGadget::TrackMouse*

MouseEnter

virtual void MouseEnter(UINT modKeys, TPoint& point);

Called when the mouse enters the gadget.

See also: *TGadget::MouseLeave*

MouseLeave

virtual void MouseLeave(UINT modKeys, TPoint& point);

Called when the mouse leaves the gadget.

See also: *TGadget::MouseEnter*

MouseMove

```
virtual void MouseMove(UINT modKeys, TPoint& point);
```

If mouse events are captured, *EvMouseMove* responds to a mouse dragging message. *point* is located in the receiver's coordinate system.

See also: *TGadget::MouseEnter*, *TGadget::MouseLeave*

Paint

```
virtual void Paint(TDC&);
```

Calls *PaintBorder* to paint the indicated device context.

See: *TTextGadget::Paint*

PaintBorder

```
virtual void PaintBorder(TDC& dc);
```

Used to paint the border, *PaintBorder* calls *::GetSystemMetrics* to obtain the width and height of the gadget and uses the color returned by *GetSyscolor* to paint or highlight the area with the specified brush. Depending on whether the border style is raised, embossed, or recessed, *PaintBorder* paints the specified boundary. You can override this function if you want to implement a border style that isn't supported by ObjectWindows's gadgets.

PtIn

```
virtual BOOL PtIn(TPoint& point);
```

PtIn determines if the point is within the receiver's bounding rectangle and returns TRUE if this is the case; otherwise, returns FALSE.

Removed

```
virtual void Removed();
```

Called after a gadget is removed from a window.

TGadgetWindow class

gadgetwi.h

Derived from *TWindow*, *TGadgetWindow* maintains a list of tiled gadgets for a window and lets you dynamically arrange tool bars. You can specify the following attributes of these gadgets:

- Horizontal or vertical tiling. Positions the gadgets horizontally or vertically within the inner rectangle (the area excluding borders and margins).
- Gadget font. Default font to use for gadgets and for calculating layout units. For font information, see the description of *TGadgetFont*.
- Left, right, top, and bottom margins. Specified in pixels, layout units (based on the window font), or border units (the width or height of a thin window border).

- Measurement units. Specified in pixels, layout units, or border units.
- Gadget window size. A gadget window can shrink-wrap its width, height, or both to fit around its gadgets. By default, horizontally tiled gadgets shrink-wrap to fit the height of the window and vertically tiled gadgets shrink-wrap to fit the width of the window.

TGadgetWindow is the base class for the following derived classes: *TControlBar*, *TMessageBar*, *TToolBox*, and *TStatusBar*.

Public constructors and destructor

Constructor

```
TGadgetWindow(TWindow *parent = 0, TTileDirection direction = Horizontal,
              TFont *font = new TGadgetWindowFont, TModule* module = 0);
```

Creates a *TGadgetWindow* interface object with the default tile direction and font and passes *module* with a default value of 0.

Destructor

```
~TGadgetWindow();
```

Destructs the *TGadgetWindow* object by deleting all of its gadgets and fonts.

Public member functions

FirstGadget

```
inline TGadget* FirstGadget() const;
```

Returns the *FirstGadget* in the list.

See also: *TGadgetWindow::FirstGadget*

GadgetChangedSize

```
void GadgetChangedSize(TGadget& gadget);
```

Used to notify the gadget window that a gadget has changed its size, *GadgetChangedSize* calls *LayoutSession* to re-layout all gadgets.

See also: *TGadget::SetShrinkWrap*, *TGadgetWindow::GadgetChangedSize*

GadgetFromPoint

```
TGadget* GadgetFromPoint(TPoint& point);
```

Returns the gadget at the given window coordinates.

GadgetReleaseCapture

```
void GadgetReleaseCapture(TGadget& gadget);
```

Releases the capture so that other windows can receive mouse messages.

See also: *TGadgetWindow::GadgetSetCapture*

GadgetSetCapture

```
BOOL GadgetSetCapture(TGadget& gadget);
```


GadgetSetCapture reserves all mouse messages for the gadget window until the capture is released. Although gadgets are always notified if a left button-down event occurs within the rectangle, the derived gadget class must call *GadgetSetCapture* if you want the gadget to be notified when a mouse drag and a mouse button-up event occurs.

See also: *TGadgetWindow::GadgetReleaseCapture*

GadgetWithId

```
TGadget *GadgetWithId(int id) const;
```

Returns a pointer to the gadget associated with the given ID (*id*).

GetDirection

```
inline TTileDirection GetDirection() const;
```

Gets the horizontal or vertical orientation of the gadgets.

See also: *TGadgetWindow::SetDirection*

GetFont

```
inline TFont &GetFont();
```

Returns the font (which is *MS Sans Serif* by default).

See also: *TGadgetWindowFont::TGadgetWindowFont*

GetFontHeight

```
inline UINT GetFontHeight() const;
```

Gets the height of the window's font.

GetHintMode

```
inline THintMode GetHintMode();
```

Returns the hint mode.

IdleAction

```
void IdleAction();
```

While no messages are waiting to be processed, *IdleAction* is called and iterates through the gadgets, invoking their *CommandEnable* member function.

See also: *TGadget::CommandEnable*

Insert

```
virtual void Insert(TGadget& gadget, TPlacement = After,
                  TGadget *sibling = 0);
```

Inserts a gadget before or after a sibling gadget (*TPlacement*). If *sibling* is 0, then the new gadget is inserted at either the beginning or the end of the gadget list. If this window has already been created, *LayoutSession* needs to be called after inserting gadgets.

See also: *TGadgetWindow::LayoutSession*, *TGadgetWindow::Remove*

LayoutSession

```
virtual void LayoutSession();
```

LayoutSession is typically called when a change occurs in the size of the margins or gadgets or when gadgets are added or deleted. *LayoutSession*

calls *TileGadgets* to tile the gadgets in the specified direction and *Invalidate* to mark the area as invalid (needs repainting).

See also: *TGadgetWindow::Insert*, *TGadgetWindow::Remove*, *TWindow::Invalidate*

NextGadget

```
inline TGadget *NextGadget(TGadget& gadget) const;
```

Returns the next gadget after *gadget* or 0 if none exists.

Remove

```
virtual TGadget* Remove(TGadget& gadget);
```

Removes a gadget from the gadget window. The gadget is returned but not destroyed. *Remove* returns 0 if the gadget is not in the window.

If this window has already been created, the calling application must call *LayoutSession* after any gadgets have been removed.

See also: *TGadgetWindow::Insert*, *TGadgetWindow::LayoutSession*

SetDirection

```
virtual void SetDirection(TTileDirection direction);
```

Sets the horizontal or vertical orientation of the gadgets. If the gadget window is already created, *SetDirection* readjusts the dimensions of the gadget window to fit around the gadgets.

See also: *TGadgetWindow::GetDirection*

SetHintCommand

```
void SetHintCommand(int id);
```

Simulates menu selection messages so that ObjectWindows command processing can display command hints.

See also: *WM_MENUSELECT*, *WM_ENTERIDLE*

SetHintMode

```
inline void SetHintMode(THintMode hintMode);
```

Sets the mode of the hint text. Defaults to *PressHints* (displays hint text when a button is pressed).

See also: *THintMode* enum

SetMargins

```
void SetMargins(TMargins &margins);
```

Sets or changes the margins for the gadget window and calls *LayoutSession*.

See also: *TGadgetWindow::Margins*

SetShrinkWrap

```
void SetShrinkWrap(BOOL shrinkWrapWidth, BOOL shrinkWrapHeight);
```

Sets the width and height of the data members. By default, if the tile direction is horizontal, *ShrinkWrapWidth* is FALSE and *ShrinkWrapHeight* is TRUE. Also by default, if the direction is vertical, *ShrinkWrapWidth* is TRUE and *ShrinkWrapHeight* is FALSE.

Protected data members

AtMouse	<p>TGadget* AtMouse;</p> <p>The last gadget at the mouse position.</p>
BkgndBrush	<p>TBrush* BkgndBrush;</p> <p>The color of the background brush.</p>
Capture	<p>TGadget *Capture;</p> <p>Points to the gadget that currently has the mouse capture; otherwise, if no gadget has the mouse capture, <i>Capture</i> is 0.</p> <p>See also: <i>TGadgetWindow::GadgetSetCapture</i></p>
Direction	<p>TTileDirection Direction;</p> <p>The direction of the tiling—either horizontal or vertical.</p>
DirtyLayout	<p>BOOL DirtyLayout;</p> <p>Indicates the layout has changed and gadgets need to be re-tiled. Using <i>DirtyLayout</i> avoids redundant tiling when gadget windows are created.</p> <p>See also: <i>TGadgetWindow::LayoutSession</i></p>
Font	<p>TFont *Font;</p> <p>Points to the font used to calculate layout units.</p> <p>See also: <i>TGadgetWindow::GetFont</i></p>
FontHeight	<p>UINT FontHeight;</p> <p>Holds the height of the gadget window's font.</p> <p>See also: <i>TGadgetWindow::GetFont</i></p>
Gadgets	<p>TGadget *Gadgets;</p> <p>Points to the first gadget in the gadget list.</p>
HintMode	<p>THintMode HintMode;</p> <p>Holds the hint text mode.</p> <p>See also: <i>THintMode</i> enum</p>
Margins	<p>TMargins Margins;</p> <p>Holds the margin values for the gadget window.</p> <p>See also: <i>TGadgetWindow::SetMargins</i></p>
NumGadgets	<p>UINT NumGadgets;</p>

The number of gadgets in the window.

ShrinkWrapHeight

BOOL ShrinkWrapHeight;

If *ShrinkWrapHeight* is TRUE, the window will shrink its width to fit the tallest gadget for horizontally tiled gadgets.

See also: *TGadgetWindow::SetShrinkWrap*

ShrinkWrapWidth

BOOL ShrinkWrapWidth;

If *ShrinkWrapWidth* is TRUE, the window will shrink its width to fit the widest gadget for vertically tiled gadgets.

See also: *TGadgetWindow::SetShrinkWrap*

WideAsPossible

UINT WideAsPossible;

The number of gadgets that are as wide as possible.

Protected member functions

Create

BOOL Create();

Overrides *TWindow member* function and chooses the initial size of the gadget if shrink-wrapping was requested.

See also: *TGadgetWindow::SetShrinkWrap*

EvLButtonDown

void EvLButtonDown(UINT modKeys, TPoint& point);

Responds to a left button-down mouse message by forwarding the event to the gadget positioned under the mouse.

EvLButtonUp

void EvLButtonUp(UINT modKeys, TPoint& point);

Responds to a left button-up mouse message by forwarding the event to the gadget that has the capture.

EvMouseMove

void EvMouseMove(UINT modKeys, TPoint& point);

If mouse events are captured, *EvMouseMove* responds to a mouse move message by forwarding the event to the gadget that has the capture.

EvSize

void EvSize(UINT sizeType, TSize &);

Calls *TWindow::EvSize* to perform any default processing. If *DirtyLayout* TRUE and *WideAsPossible* is greater than 0, *EvSize* sets *DirtyLayout* to TRUE and calls *TileGadgets* to readjust the size and *Invalidate* to mark the area for redrawing.

See also: *TWindow::EvSize*

EvSysColorChange void EvSysColorChange();

EvSysColorChange, which is called when any system colors have changed, forwards the event to all gadgets.

GetDesiredSize virtual void GetDesiredSize(TSize &size);

If shrink-wrapping was requested, *GetDesiredSize* returns the size needed to accommodate the borders and the margins of the widest and highest gadget; otherwise, it returns the width and height in the window's *Attr* structure.

If you want to leave extra room for a specific look (for example, a separator line between gadgets, a raised line, and so on), you can override this function. However, if you override *GetDesiredSize*, you will probably also need to override *GetInnerRect* to calculate your custom inner rectangle.

See also: *TGadgetWindow::GetInnerRect*

GetInnerRect virtual void GetInnerRect(TRect &);

GetInnerRect computes the rectangle inside of the borders and margins of the gadget.

If you want to leave extra room for a specific look (for example, a separator line between gadgets, a raised line, and so on), you can override this function. If you override *GetInnerRect*, you will probably also need to override *GetDesiredSize* to calculate your custom total size.

See also: *TGadgetWindow::GetDesiredSize*

GetMargins void GetMargins(TMargins&, int &left, int &right, int &top, int &bottom);

Returns the left, right, top, and bottom margins in pixels.

LayoutUnitsToPixels int LayoutUnitsToPixels(int units);

Converts layout units to pixels. A layout unit is determined by dividing the window font height by eight.

See also: *TGadgetWindow::LayoutSession*

Paint void Paint(TDC& dc, BOOL erase, TRect& rect);

Puts the font into the device context and calls *PaintGadgets*.

See also: *TGadgetWindow::PaintGadgets*

PaintGadgets virtual void PaintGadgets(TDC& dc, BOOL erase, TRect& rect);

Called by *Paint* to repaint all of the gadgets, *PaintGadgets* iterates through the list of gadgets, determines the gadget's area, and repaints each gadget.

You can override this function to implement a specific look (for example, separator line, raised, and so on).

See also: *TGadgetWindow::EvPaint*

PositionGadget

```
virtual void PositionGadget(TGadget *previous, TGadget *next, TPoint&
                             point);
```

PositionGadget is called to allow spacing adjustments to be made before each gadget is positioned.

See also: *TGadgetWindow::TileGadgets*

TileGadgets

```
virtual void TileGadgets();
```

Tiles the gadgets in the direction requested (horizontal or vertical). Calls *PositionGadget* to give derived classes an opportunity to adjust the spacing between gadgets in their windows.

See also: *TGadgetWindow::PositionGadget*

Response table entries

Response table entry	Member function
EV_WM_LBUTTONDOWN	EvLButtonDown
EV_WM_LBUTTONUP	EvLButtonUp
EV_WM_MOUSEMOVE	EvMouseMove
EV_WM_SIZE	EvSize
EV_WM_SYSCOLORCHANGE	EvSysColorChange

TGadgetWindowFont class

gadgetwi.h

Derived from *TFont*, *TGadgetWindowFont* is a specific font used in gadget windows for sizing and default text. You can specify the point size of the font (not the size in pixels) and whether it is bold or italic. You can use a variety of the Windows API FW_xxxx or FF_xxxx constants to set the font type.

Public constructors

Constructor

```
TGadgetWindowFont(int pointSize = 10, BOOL bold = FALSE,
                  BOOL italic = FALSE);
```

Constructs a *TGadgetWindowFont* interface object with a default point size of 10 picas without bold or italic typeface. By default, the constructor creates the system font: a variable-width, sans-serif Helvetica.

TGauge class

gauge.h

A streamable class derived from *TControl*, *TGauge* defines the basic behavior of gauge controls. Gauges are display-only horizontal or vertical controls that provide duration or analog information about a particular process. In general, horizontal gauges with a broken (dashed-line) bar are used to display short-duration, process information whereas horizontal gauges with a solid bar are used to illustrate long-duration, process information. Usually, vertical gauges are preferred for displaying analog information.

Public member functions

GetRange

```
inline void GetRange(int& min, int& max) const
```

Gets the minimum and maximum values for the gauge.

GetValue

```
inline int GetValue() const
```

Gets the current value of the gauge.

SetLed

```
void SetLed(int spacing, int thick = 90);
```

Sets the *LedSpacing* and *LedThick* data members to the values *spacing* and *thick*.

SetRange

```
void SetRange(int min, int max);
```

Sets the *Min* and *Max* data members to *min* and *max* values returned by the constructor. If *Max* is less than or equal to *Min*, *SetRange* resets *Max* to *Min* plus 1.

SetValue

```
void SetValue(int value);
```

Restricts the value so that it is within the minimum and maximum values established for the gauge. If the current value has changed, *SetValue* marks the old position for repainting. Then, it sets the data member *Value* to the new value.

Public constructors

Constructor

```
TGauge(TWindow* parent, const char far* title, int id,
        int X, int Y, int W, int H, BOOL isHorizontal = TRUE,
        int margin = 0, TModule* module = 0);
```

Constructs a *TGauge* object with borders that are determined by using the value of `SM_CXBORDER` from `::GetSystemMetrics`. Sets *IsHorizontal* to `isHorizontal` Sets border thickness and spacing between dashed borders (LEDs) to 0. Sets the range of possible values from 0 to 100.

Protected data members

IsHorizontal

```
int IsHorizontal;
```

Set to the *ishorizontal* argument of the constructor. *IsHorizontal* is TRUE if the gauge is horizontal and FALSE if it is vertical.

LedSpacing

```
int LedSpacing;
```

Holds the integer value (in gauge units) of the spacing between the broken bars of the gauge.

LedThick

```
int LedThick;
```

Holds the thickness of the broken bar.

Max

```
int Max;
```

Holds the maximum value (in gauge units) displayed on the gauge.

Min

```
int Min;
```

Holds the minimum value (in gauge units) displayed on the gauge.

Margin

```
int Margin;
```

The border width and height of the gauge. *Margin* is calculated by multiplying the value returned by `GetSystemMetrics` in the `SM_CXBORDER` parameter by the value in the constructor's parameter, *margin*.

Value

```
int Value;
```

Holds the current value of the gauge.

Protected member functions

EvEraseBkgnd

```
BOOL EvEraseBkgnd(HDC);
```


Overrides *TWindow's EvEraseBkgnd* function and erases the background of the gauge. Whenever the background is repainted, *EvEraseBkgnd* is called to avoid flickering.

See also: *TWindow::EvEraseBkgnd*

Paint

```
void Paint();
```

Overrides *TWindow's Paint* function and paints the area and border of the gauge. Assigns the horizontal and vertical border values of the gauge's rectangle to the values of the *SM_CXBORDER* and *SM_CYBORDER* parameters of *GetSystemMetrics*. Determines the color by using the value returned by the *COLOR_BTNshadow* option of *GetSysColor* and uses the values in *LedSpacing* and *IsHorizontal* to draw a horizontal or vertical gauge with solid or broken bars.

See also: *TGauge::LedSpacing, TGauge::IsHorizontal*

Response table entries

Response table entry	Member function
EV_WM_ERASEBKGND	EvEraseBkgnd

TGdiObject class

gdiobjec.h

GdiObject is the root, pseudo-abstract base class for *ObjectWindows's* GDI (Graphics Device Interface) wrappers. WIN API calls that take a GDI handle argument are typically replaced by simpler *ObjectWindows* member function invocations in which the handle (and possibly other arguments) is "supplied" by the calling object. The *TGdiObject*-based classes let you work with a GDI handle and construct a C++ object with an aliased handle. Some GDI objects are also based on *TGdiObject* for handle management. Generally, the *TGdiObject*-based class hierarchy handles all GDI objects apart from the DC (Device Context) objects handled by the *TDC*-based tree.

The five DC selectable classes (*TPen, TBrush, TFont, TPalette, and TBitmap*), and the *TIcon, TCursor, TDib, and TRegion* classes, are all derived directly from *TGdiObject*.

TGdiObject maintains the GDI handle and a *ShouldDelete* flag that determines if and when the handle and object should be destroyed.

Protected constructors are provided for use by the derived classes: one for borrowed handles, and one for normal use.

An optional orphan control mechanism is provided. By default, orphan control is active, but you can turn it off by defining the `NO_GDI_ORPHAN_CONTROL` identifier:

```
#define NO_GDI_ORPHAN_CONTROL
```

With orphan control active, the following static member functions are available:

RefAdd, *RefCount*, *RefDec*, *RefFind*, *RefInc*, and *RefRemove*.

These maintain object reference counts and allow safe orphan recovery and deletion. Macros, such as `OBJ_REF_ADD`, let you deactivate or activate your orphan control code by simply defining or undefining `NO_GDI_ORPHAN_CONTROL`. When `NO_GDI_ORPHAN_CONTROL` is undefined, for example, `OBJ_REF_ADD(handle, type)` expands to `TGdiObject::RefAdd((handle),(type))`, but when `NO_GDI_ORPHAN_CONTROL` is defined, the macro expands to `handle`.

Public data members

enum TAutoDelete

```
enum TAutoDelete(NoAutoDelete, AutoDelete);
```

Enumerates the flag values for GDI Handle constructors. This flag is used to control GDI object deletion in the destructors.

See also: The second protected constructor in this class

enum TType

```
enum TType(TpNone, TpPen, TpBrush, TpFont, TpPalette, TpBitmap,
           TpTextBrush);
```

This enumeration is used to store the object type in the **struct** `TObjInfo`. This internal structure is used to track object reference counts during debugging sessions.

See also: `TGdiObject::RefXXX`

Public member functions

GetObject

```
inline int GetObject(int count, void far* object) const;
```

Wrapper for the Win API `GetObject(Handle, count, object)` call. Obtains information about this GDI object and places it in the `object` buffer. If the call succeeds and `object` is not 0, `GetObject` returns the number of bytes copied to the object buffer. If the call succeeds and `object` is 0, `GetObject` returns the

number of bytes needed in the object buffer for the type of object being queried.

See also: `::GetObject`, `TPen::GetObject`

IsGDIObject



```
inline BOOL IsGDIObject();
```

Returns TRUE if the data member *Handle* represents an existing (valid) GDI object.

See also: `TGdiObject::IsOK`, `TGdiObject::Handle`

IsOk

```
inline BOOL IsOK() const;
```

Returns TRUE if the current *Handle* is nonzero.

See also: `TGdiObject::Handle`

RefAdd

```
static void RefAdd(HANDLE handle, TType type);
```

Available only if orphan control is active (that is, if `NO_GDI_ORPHAN_CONTROL` is undefined). *RefAdd* adds a reference entry for the object with the given *handle* and *type* to the *ObjInfoBag* table and sets the reference count to 1. If the table already has a matching entry, no action is taken.

See also: `TGdiObject::RefXXX`, macro `OBJ_REF_ADD`

RefCount

```
static int RefCount(HANDLE handle);
```

Available only if orphan control is active, that is, if `NO_GDI_ORPHAN_CONTROL` is undefined. *RefCount* returns this object's current reference count or -1 if the object is not in the *ObjInfoBag* table.

See also: `TGdiObject::RefXXX`, macro `OBJ_REF_COUNT`

RefDec

```
static void RefDec(HANDLE handle);
static void RefDec(HANDLE handle, BOOL wantDelete);
```

Available only if orphan control is active, that is, if `NO_GDI_ORPHAN_CONTROL` is undefined. *RefDec* decrements this object's reference count by 1 and deletes the object when the reference count reaches zero. A warning is issued if the deletion was supposed to happen but didn't. Likewise, a warning is issued if the deletion wasn't supposed to happen but did. The deleted object is also detached from the *ObjInfoBag* table.

The second version of *RefDec* is available only if the `__TRACE` identifier is defined. You can vary the normal deletion strategy by setting *wantDelete* to TRUE or FALSE.

See also: `TGdiObject::RefXXX`, macro `OBJ_REF_DEC`

RefFind static TObjInfo* RefFind(HANDLE object);

Available only if orphan control is active (that is, if NO_GDI_ORPHAN_CONTROL is undefined). *RefFind* searches the *ObjInfoBag* table for an entry for the given object. If found, the object's type and reference count are returned in the specified *TObjInfo* object. *RefFind* returns 0 if no match is found.

See also: *TGdiObject::RefXXX*, *TObjInfo*, *TObjInfoBag*

RefInc static void RefInc(HANDLE handle);

Available only if orphan control is active (that is, if NO_GDI_ORPHAN_CONTROL is undefined). *RefInc* increments by 1 the reference count of the object associated with *handle*.

See also: *TGdiObject::RefXXX*, macro *OBJ_REF_INC*

RefRemove static void RefRemove(HANDLE handle);

Available only if orphan control is active (that is, if NO_GDI_ORPHAN_CONTROL is undefined). *RefRemove* removes the reference entry to the object with the given *handle* from the *ObjInfoBag* table. If the given handle is not found, no action is taken.

See also: *TGdiObject::RefXXX*, macro *OBJ_REF_REMOVE*

Protected data members

Handle HANDLE Handle;

The GDI handle of this object.

See also: *TGdiObject* constructors

ShouldDelete BOOL ShouldDelete;

Set TRUE if the destructor needs to delete this object's GDI handle.

See also: *TGdiObject* constructors and destructor

Protected constructors and destructor

Constructor TGdiObject();

This default constructor sets *Handle* to 0 and *ShouldDelete* to TRUE. This constructor is intended for use by derived classes that must set the *Handle* member.

See also: *TGdiObject::Handle*, *TGdiObject::ShouldDelete*

Constructor

```
TGdiObject(HANDLE handle, TAutoDelete autoDelete = NoAutoDelete);
```

This constructor is intended for use by derived classes only. The *Handle* data member is “borrowed” from an existing handle, given by the argument *handle*. The *ShouldDelete* data member defaults to FALSE, ensuring that the borrowed handle will not be deleted when the object is destroyed.

See also: *enum TAutoDelete*, *TGdiObject::ShouldDelete*

Destructor

```
virtual ~TGdiObject();
```

If *ShouldDelete* is FALSE, no action is taken. Otherwise, with *ShouldDelete* TRUE, the action of the destructor depends on whether orphan control is active or not. If orphan control is inactive (that is, if NO_ORPHAN_CONTROL is defined) *~TGdiObject* deletes the GDI object by calling *::DeleteObject(Handle)*. If orphan control is active (the default), *~TGdiObject* calls *RefDec(Handle, TRUE)*, so that *::DeleteObject(Handle)* is called only if the object’s reference count is 0.

See also: *TGdiObject::Handle*, *TGdiObject::ShouldDelete*, *TGdiObject::RefXXX*

Macros

OBJ_REF_ADD

```
OBJ_REF_ADD(handle, type)
```

If orphan control is active (the default), *OBJ_REF_ADD(handle, type)* is defined as *TGdiObject::RefAdd((handle), (type))*. The latter adds to the *ObjInfoBag* table a reference entry for the object with the given *handle* and *type*, and sets its count to 1.

If orphan control is inactive, *OBJ_REF_ADD(handle)* is defined as *handle*. This macro lets you write orphan control code that can be easily deactivated with the single statement `#define NO_GDI_ORPHAN_CONTROL`.

See also: *TGdiObject::RefAdd*

OBJ_REF_COUNT

```
OBJ_REF_COUNT(handle)
```

If orphan control is active (the default), *OBJ_REF_COUNT(handle)* is defined as *TGdiObject::RefCount((handle))*. The latter returns the reference count of the object with the given handle, or -1 if no such object exists. If orphan control is inactive, *OBJ_REF_COUNT(handle)* is defined as -1. This macro lets you write orphan control code that can be easily deactivated with the single statement `#define NO_GDI_ORPHAN_CONTROL`.

See also: *TGdiObject::RefCount*

OBJ_REF_DEC

OBJ_REF_DEC(handle, wantDelete)

If orphan control is active (the default), *OBJ_REF_DEC*(handle, wantDelete) is defined as either *TGdiObject::RefDec*((handle)) or *TGdiObject::RefDec*((handle), (wantDelete)). The latter format occurs only if `_TRACE` is defined.

RefDec(handle) decrements the reference count of the object associated with *handle* and optionally deletes orphans or warns you of their existence. If orphan control is inactive, *OBJ_REF_DEC*(handle) is defined as *handle*. This macro lets you write orphan control code that can be easily deactivated with the single statement `#define NO_GDI_ORPHAN_CONTROL`.

See also: *TGdiObject::RefDec*

OBJ_REF_INC

OBJ_REF_INC(handle)

If orphan control is active (the default), *OBJ_REF_INC*(handle) is defined as *TGdiObject::RefInc*((handle)). The latter increments the reference count of the object associated with *handle*. If orphan control is inactive, *OBJ_REF_DEC*(handle) is defined as *handle*. This macro lets you write orphan control code that can be easily deactivated with the single statement `#define NO_GDI_ORPHAN_CONTROL`.

See also: *TGdiObject::RefInc*

OBJ_REF_REMOVE

OBJ_REF_REMOVE(handle)

If orphan control is active (the default), *OBJ_REF_REMOVE*(handle) is defined as *TGdiObject::RefRemove*((handle)). The latter removes from the *ObjInfoBag* table the reference entry for the object associated with *handle*. If orphan control is inactive, *OBJ_REF_REMOVE*(handle) is defined as *handle*. This macro lets you write orphan control code that can be easily deactivated with the single statement `#define NO_GDI_ORPHAN_CONTROL`.

See also: *TGdiObject::RefRemove*

TGdiObject::TXGdi class**gdibase.h**

Describes an exception resulting from GDI failures such as creating too many TWindow DCs. This exception occurs, for example, if a dc driver can't be located or if a DIB file can't be read.

The following code from the PAINT.CPP sample program on your distribution disk throws a TXGdi exception if a new DIB can't be created.

```

void TCanvas::NewDib(int width, int height, int nColors)
{
    TDib* dib;
    try {
        dib = new TDib(width, height, nColors);
    }
    catch (TGdiObject::TXGdi& x) {
        MessageBox("Could Not Create DIB", GetApplication()->Name, MB_OK);
        return;
    }
}

```

Public constructors

Constructor

```
TXGdi(UINT resId = IDS_GDIFAILURE, HANDLE = 0);
```

Constructs a *TXGdi* object with a default IDS_GDIFAILURE message.

Public member functions

Msg

```
static string Msg(UINT resId, HANDLE);
```

Converts the resource ID to a string and returns the string message.

TGroupBox class

groupbox.h

An instance of a *TGroupBox* is an interface object that represents a corresponding group box element in Windows. Generally, *TGroupBox* objects are not used in dialog boxes or dialog windows (*TDialog*), but are used when you want to create a group box in a window.

Although group boxes don't serve an active purpose onscreen, they visually unify a group of selection boxes such as check boxes and radio buttons or other controls. Behind the scenes, however, they can take an important role in handling state changes for their group of controls (normally check boxes or radio buttons).

For example, you might want to respond to a selection change in any one of a group of radio buttons in a similar manner. You can do this by deriving a class from *TGroupBox* that redefines the member function *SelectionChanged*.

Alternatively, you could respond to selection changes in the group of radio buttons by defining a response for the group box's parent. To do so, define a child-ID-based response member function using the ID of the group box. The group box will automatically send a child-ID-based message to its

parent whenever the radio button selection state changes. This class is streamable.

Public data members

NotifyParent

```
BOOL NotifyParent;
```

Flag that indicates whether parent is to be notified when the state of the group box's selection boxes has changed. *NotifyParent* is TRUE by default.

Public constructors

Constructor

```
TGroupBox(TWindow* parent, int Id, const char far *text, int x, int y,
          int w, int h, TModule* module = 0);
```

Constructs a group box object with the supplied parent window (*Parent*), control ID (*Id*), associated text (*text*), position (*x, y*) relative to the origin of the parent window's client area, width (*w*), and height (*h*). Invokes the *TControl* constructor with similar parameters, then modifies *Attr.Style*, adding *BS_GROUPBOX* and removing *WS_TABSTOP*. *NotifyParent* is set to TRUE; by default, the group box's parent is notified when a selection change occurs in any of the group box's controls.

See also: *TControl::TControl*

Constructor

```
TGroupBox(TWindow* parent, int resourceId, TModule* module = 0);
```

Constructs a *TGroupBox* object to be associated with a group box control of a *TDialog*. Invokes the *TControl* constructor with identical parameters. *resourceID* must correspond to a group box resource that you define.

See also: *TControl::TControl*, *TWindow::DisableTransfer*

Public member functions

SelectionChanged

```
virtual void SelectionChanged(int controlId);
```

If *NotifyParent* is TRUE, *SelectionChanged* notifies the parent window of the group box that one of its selections has changed by sending it a child-ID-based message. This member function can be redefined to allow the group box to handle selection changes in its group of controls.

Protected member functions

GetClassName

```
char far* GetClassName();
```

GetClassName returns the name of *TGroupBox*'s Windows registration class, "BUTTON." If BWCC is enabled, *GetClassName* returns `BUTTON_CLASS`.

THintMode enum

gadgetwi.h

Enumerates the hint mode settings of the gadget—either no hints, hints when a button is pressed, or hints when the mouse passes over a gadget.

```
enum THintMode
```

See also: *TGadgetWindow::GetHintMode*

THSlider class

slider.h

Derived from *TSlider*, *THSlider* provides implementation details for horizontal sliders. See *TSlider* for an illustration of a horizontal slider.

Public constructors

Constructor

```
THSlider(TWindow* parent, int id, int X, int Y, int W, int H,  
         TResId thumbResId, TModule* module = 0);
```

Constructs a slider object.

Protected member functions

HitTest

```
int HitTest(TPoint& point);
```

Overrides *TSlider*'s virtual function and gets information about where a given X, Y location falls on the slider. The return value is in *scrollCodes*.

See also: *TSlider::HitTest*

NotifyParent

```
void NotifyParent(int scrollCode, int pos=0);
```

Overrides *TSlider*'s virtual function and sends a `WS_HSCROLL` message to the parent window.

See also: *TSlider::NotifyParent*

PaintRuler	<pre>void PaintRuler(TDC&);</pre> <p>Overrides <i>TSlider's</i> virtual function and paints the horizontal ruler.</p> <p>See also: <i>TSlider::PaintRuler</i></p>
PaintSlot	<pre>void PaintSlot(TDC&);</pre> <p>Overrides <i>TSlider's</i> virtual function and paints the slot in which the thumb slides.</p> <p>See also: <i>TSlider::PaintSlot</i></p>
PointToPos	<pre>int PointToPos(TPoint& point);</pre> <p>Overrides <i>TSlider's</i> virtual function and translates an X,Y point to a position in slider units.</p> <p>See also: <i>TSlider::PointToPos</i></p>
PosToPoint	<pre>TPoint PosToPoint(int pos);</pre> <p>Overrides <i>TSlider's</i> virtual function and translates a position in slider units to an X,Y point.</p> <p>See also: <i>TSlider::PosToPoint</i></p>

TIC class**dc.h**

Derived from *TDC*, *TIC* is a DC class that provides a constructor for creating a DC object from explicit driver, device, and port names.

Public constructors**Constructor**

```
TIC(const char far* driver, const char far* device, const char far*
    output, const DEVMODE far* initData=0);
```

Creates a DC object with the given driver, device, and port names and initialization values.

See also: *::DeviceCapabilitiesEx*, *DEVMODE* struct, *TDC::GetDeviceCaps*

TIcon class**gdiobjec.h**

TIcon, derived from *TGdiobject*, represents the GDI object icon class. *TIcon* constructors can create icons from a resource or from explicit information.

Because icons are not real GDI objects, the *TIcon* destructor overloads the base destructor, *~TGdiObject()*.

Public constructors and destructor

Constructor

```
TIcon(HICON handle, TAutoDelete autoDelete = NoAutoDelete);
```

Creates a *TIcon* object and sets the *Handle* data member to the given borrowed *handle*. The *ShouldDelete* data member defaults to *FALSE*, ensuring that the borrowed handle will not be deleted when the C++ object is destroyed.

See also: *TGdiObject::Handle*, *TGdiObject::ShouldDelete*,

Constructor

```
TIcon(HINSTANCE instance, const TIcon& icon);
```

Creates a copy of the given *icon* object by calling the Win API function *CopyIcon(instance, icon)*.

See also: *::CopyIcon*

Constructor

```
TIcon(HINSTANCE instance, TResID resID);
```

Creates an icon object from the given resource.

See also: *::LoadIcon*

Constructor

```
TIcon(HINSTANCE instance, const char far* filename, int index);
```

Creates an icon object from the given resource file.

See also: *::ExtractIcon*

Constructor

```
TIcon(HINSTANCE instance, const TSize& size, int planes, int bitsPixel,
      const void far* andBits, const void far* xorBits);
```

Creates an icon object with the given values.

See also: *::CreateIcon*

Constructor

```
TIcon(const void* resBits, DWORD resSize);
```

Creates an icon object of the given size from the bits found in the *resBits* buffer.

See also: *::CreateIconFromResource*

Constructor

```
TIcon(const ICONINFO* iconInfo);
```

Creates an icon object with the given *ICONINFO* information.

See also: *::CreateIconIndirect*




Destructor `~TIcon();`

Overrides the base destructor to call `::DestroyIcon` instead of the default `::DeleteObject`.

See also: `~TGdiObject`, `::DestroyIcon`

Public member functions


GetIconInfo `inline BOOL GetIconInfo(ICONINFO* iconInfo) const;`



Retrieves information about this icon and copies it in the given `ICONINFO` structure. Returns `TRUE` if the call is successful; otherwise returns `FALSE`.

See also: `::GetIconInfo`, `struct ICONINFO`

operator HICON() `inline operator HICON() const;`



Typecasting operator that converts this icon's *Handle* to type *HICON* (the Windows data type representing the handle to an icon resource).

TInputDialog class

inputdia.h

TInputDialog provides a generic dialog box to retrieve text input by a user. When the input dialog box is constructed, its title, prompt, and default input text are specified. *TInputDialog* is a streamable class.

Public data members

buffer `char far* buffer;`

Pointer to the buffer that returns the text retrieved from the user. When passed to the constructor of the input dialog box, contains the default text to be initially displayed in the edit control.

BufferSize `int BufferSize;`

Contains the size of the buffer that returns user input.

prompt `char far* prompt;`

Points to the prompt for the input dialog box.

Public constructors

Constructor

```
TInputDialog(TWindow* parent, const char far *title,
             const char far *prompt, char far* buffer, int bufferSize,
             TModule* module = 0);
```

Invokes *TDialog*'s constructor, passing it *parent*, the resource identifier and *module*. Sets the caption of the dialog box to *title* and the prompt static control to *prompt*. Sets the *Buffer* and *BufferSize* data members to *buffer* and *bufferSize*.

See also: *TDialog::TDialog*

Public member function

TransferData

```
void TransferData(TTransferDirection direction);
```

Transfers the data of the input dialog box. If *direction* is *tdSetData*, sets the text of the static and edit controls of the dialog box to the text in *prompt* and *buffer*. If *direction* is *tdGetData*, fills the *buffer* with the current text of the *Editor*.

Protected member function

SetupWindow

```
virtual void SetupWindow();
```

In setting up the window, *SetupWindow* calls *TDialog::SetupWindow*, then limits the number of characters the user can enter to *bufferSize* - 1.

TInStream class

docview.h

Derived from *TStream* and *istream*, *TInStream* is a base class used for defining input streams for documents.

Public constructors

Constructor

```
TInStream(TDocument& doc, LPCSTR name, int mode);
```

Constructs a *TInStream* object. *doc* refers to the document object, *name* is the user-defined name of the stream, and *mode* is the mode of opening the stream.

See also: *TOutputStream*, ofXXXX document open enum, shdocument sharing enum

TKeyboardModeTracker class

keymodet.h

Note: The functionality of this class is provided in *TStatusBar*. This class no longer exists.

Derived from *TWindow*, *TKeyboardModeTracker* is a streamable, mix-in class designed to track changes in keyboard modes that occur when the toggle keys (*CapsLock*, *NumLock*, *ScrollLock*), or edit keys (*Ins*) keys are pressed. By setting *updateStatusBar* in the constructor to **TRUE**, you can also specify that the class updates the status bar whenever a mode change occurs.

Public data members

TModeIndicator

enum TModeIndicator

A subtype of the mode indicators supported by *TStatusBar*, *TKeyboardModeTracker*'s mode indicator sets the *CapsLock*, *NumLock*, *ScrollLock*, and *Overtyp*e indicators equivalent to those of *TStatusBar*'s mode indicators.

See also: *TStatusBar::TModeIndicator*

Public constructors

Constructor

```
TKeyboardModeTracker(TWindow* parent, BOOL updateStatusBar = TRUE, UINT
    modes = CapsLock | NumLock | Overtyp, TModule*
    module = 0);
```

Constructs a *TKeyboardModeTracker* object that keeps track of the keyboard modes and updates the status bar if *updateStatusBar* is set to **TRUE** (the default value). You can use the *modes* parameter to indicate which modes you want your application to track.

Protected data members

Modes

UINT Modes;

One or more of the keyboard indicator modes that you want your program to track. The indicator modes correspond to the following virtual key codes:

Mode	Virtual key code
CapsLock	VK_CAPITAL
NumLock	VK_NUMLOCK
Overtyping	VK_INSERT
ScrollLock	VK_SCROLL

OvertypingState

BOOL OvertypingState;

Is TRUE if overtyping mode is activated.

ScrollLockState

BOOL ScrollLockState

Is TRUE if scroll lock mode is activated.

UpdateStatusBar

BOOL UpdateStatusBar;

Is TRUE if changes in keyboard modes are to be reflected on the status bar.

See also: *TStatusBar::TStatusBar*

Protected member functions**DoKeyDown**

TEventStatus DoKeyDown(UINT key, UINT repeatCount, UINT flags);

DoKeyDown translates the virtual key code into a movement, calls *TStatusBar*'s member function to change the status bar's mode indicator, and then returns *TEventStatus*. *key* indicates the virtual key code of the pressed key, *repeatCount* holds the number of times the same key is pressed, *flags* contains one of the messages that translates to a virtual key (VK) code for the mode indicators. If *UpdateStatusBar* is TRUE and the state of the *Ins* key or *ScrollLock* key changes, *DoKeyDown* passes the current state of the keyboard to *TStatusBar::SetModeIndicator*, which updates the mode indicators on the status bar. If *UpdateStatusBar* is TRUE and the state of the *NumLock* key or *CapsLock* key changes, *DoKeyDown* passes the current state of the keys to *TStatusBar::ToggleModeIndicator*, which updates the mode indicators on the status bar.

See also: *TEventStatus* enum, *TKeyboardModeTracker::EvKeyDown*

DoSetFocus

TEventStatus DoSetFocus(HWND hWndLostFocus);

Restores the focus to the active window and returns *TEventStatus*. *hWndLostFocus* contains a handle to the window that lost the focus. If *UpdateStatusBar* is TRUE, *DoSetFocus* determines if any changes have occurred in the keyboard mode and, if appropriate, updates the status of the mode indicator on the status bar by calling *TStatusBar::SetModeIndicator*.

See also: *TEventStatus* enum, *TKeyboardModeTracker::EvSetFocus*

EvKeyDown

```
void EvKeyDown(UINT key, UINT repeatCount, UINT flags);
```

Responds to a key-down message by calling *DoKeyDown*. If *DoKeyDown* doesn't return *IsComplete*, *EvKeyDown* calls *TWindow::EvKeyDown*.

See also: *TKeyboardModeTracker::DoKeyDown*

EvSetFocus

```
void EvSetFocus(HWND hWndLostFocus);
```

Responds to a set-focus message by calling *DoSetFocus*. If *DoSetFocus* doesn't return *IsComplete*, *EvSetFocus* calls *TWindow::EvSetFocus*.

See also: *TKeyboardModeTracker::DoSetFocus*

OvertypemodeChange virtual void OvertypemodeChange(BOOL on);

If overtyping is changed from on to off or vice versa, *OvertypemodeChange* is set to TRUE.

ScrollLockModeChange virtual void ScrollLockModeChange(BOOL on);

If scroll locking is changed from on to off or vice versa, *ScrollLockModeChange* is set to TRUE.

Response table entries

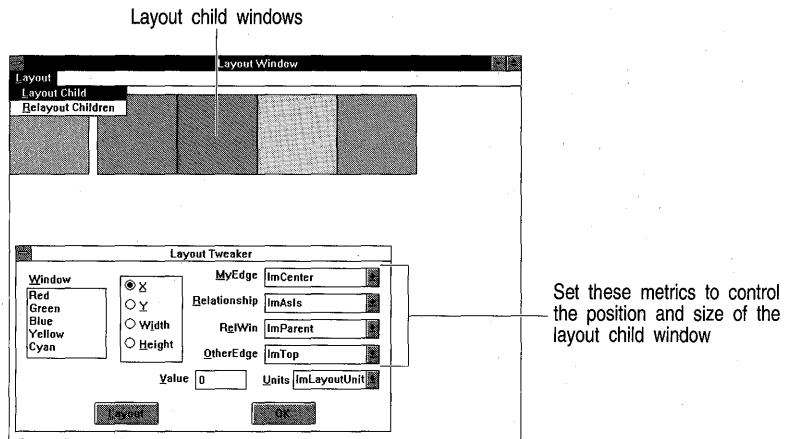
Response table entry	Member function
EV_WM_KEYDOWN	EvKeyDown
EV_WM_SETFOCUS	EvSetFocus

TLayoutConstraint struct

layoutco.h

TLayoutConstraint is a structure that defines layout constraints. Layout constraints are specified as a relationship between an edge or size of one window and an edge or size of one of the window's siblings or its parent. If a parent-child relationship is established between windows, the dimensions of the child windows are dependent on the parent window. A window can have one of its sizes depend on the size of the opposite dimension. For example, the width can be twice the height. *TLayoutMetrics* lists the relationships you can have among size and edge constraints

LAYOUT.CPP in the OWLAPI\LAYOUT directory shows you the following example of how to set up layout constraints.



Public data members

MyEdge

```
UINT MyEdge;
```

MyEdge contains the name of the edge or size constraint (*lmTop*, *lmBottom*, *lmLeft*, *lmRight*, *lmCenter*, *lmWidth*, or *lmHeight*) for your window.

See also: *TWidthHeight enum*

OtherEdge

```
UINT OtherEdge;
```

OtherEdge contains the *MyEdge* contains the name of the edge or size constraint (*lmTop*, *lmBottom*, *lmLeft*, *lmRight*, *lmCenter*, *lmWidth*, or *lmHeight*) for the other window.

See also: *TWidthHeight enum*

Relationship

```
TRelationship Relationship;
```

Relationship specifies the type of relationship that exists between the two windows (that is, *lmRightOf*, *lmLeftOf*, *lmAbove*, *lmBelow*, *lmSameAs*, or *lmPercentOf*). A value of *lmAbsolute* actually indicates that no relationship exists.

See also: *TRelationship enum*

RelWin

```
TWindow *RelWin;
```

RelWin is a pointer to the sibling windows or *lmParent* if the child is a proportion of the parent's dimensions. *RelWin* points to the window itself (**this**) if a child window's dimension is a proportion of one of its other dimensions (for example, its height is a proportion of its width).

See also: *TRelationship enum*

Units

TMeasurementUnits Units;

Units enumerates the units of measurement (either pixels or layout units) used to measure the height and width of the windows. Unlike pixels, layout units are based on system font size and will be consistent in their perceived size even if the screen resolution changes.

See also: *TMeasurementUnits enum*

union TLayoutConstraint

```
union {
    int Margin;
    int Value;
    int Percent;
};
```

This union is included for the convenience of naming the layout constraints. *Margin* is used for *ImAbove*, *ImLeftOf*, *ImLeftOf*, or *ImRightOf* enumerated values in *TRelationship*. *Value* is used for *ImSameAs* or *ImAbsolute* enumerated value in *TRelationship*. *Percent* is used for the *ImPercentOf* enumerated value in *TRelationship*.

See also: *TMeasurementUnits enum*

TLayoutMetrics class

layoutwi.h

TLayoutMetrics contains the four layout constraints used to define the layout metrics for a window. This table lists the constraints you can use for the *X*, *Y*, *Height*, and *Width* fields.

Field	Constraints
X	ImLeft, ImCenter, ImRight
Y	ImTop, ImCenter, ImBottom
Height	ImCenter, ImRight, ImWidth
Width	ImCenter, ImBottom, ImHeight

If the metrics for the child window are relative to the parent window, the relation window pointer (*ImParent*) needs to be *ImParent* (not the actual parent window pointer). For example,

```
TWindow* child = new TWindow(this, "");
TLayoutMetrics metrics;
metrics.X.Set(ImCenter, ImSameAs, ImParent, ImCenter);
metrics.Y.Set(ImCenter, ImSameAs, ImParent, ImCenter);
SetChildLayoutMetrics(*child, metrics);
```

The parent window pointer (**this**) should not be used as the relation window pointer of the child window.

Public data members

Height

TEdgeOrWidthConstraint Height;

Contains the height size constraint, center edge, or bottom edge constraint of the window.

Width

TEdgeOrWidthConstraint Width;

Contains the width size constraint, center edge, or right edge (*lmRight*) constraint of the window.

X, Y

TEdgeConstraint X, Y;

X contains the X (left, center, right) edge constraint of the window. Y contains the Y (top, center, bottom) edge constraint of the window.

Public constructors

Constructor

TLayoutMetrics();

Creates a *TLayoutMetrics* object and initializes the object by setting the units for the child and parent window to the specified layout units and the relationship between the two windows to what is defined in *lmAsIs* (of *TRelationship*). Sets the following default values:

```
X.RelWin = 0;
X.MyEdge = lmLeft;
X.Relationship = lmAsIs;
X.Units = lmLayoutUnits;
X.Value = 0;
Y.RelWin = 0;
Y.MyEdge = lmTop;
Y.Relationship = lmAsIs;
Y.Units = lmLayoutUnits;
Y.Value = 0;
Width.RelWin = 0;
Width.MyEdge = lmWidth;
Width.Relationship = lmAsIs;
Width.Units = lmLayoutUnits;
Width.Value = 0;
Height.RelWin = 0;
Height.MyEdge = lmHeight;
Height.Relationship = lmAsIs;
```

```
Height.Units = lmLayoutUnits;
Height.Value = 0;
```

The following program creates two child windows and a frame into which you can add layout constraints.

```
#include <owl\owl.h>
#include <owl\framewin.h>
#include <owl\applicat.h>
#include <owl\layoutwi.h>
#include <owl\decorate.h>
#include <owl\decmdifr.h>
#include <owl\layoutco.h>
#pragma hdrstop

// Create a derived class. //

class TMyDecoratedFrame : public TDecoratedFrame {
public:
    TMyDecoratedFrame(TWindow* parent, const char far* title,
                     TWindow& clientWnd, TWindow* MyChildWindow);
    void SetupWindow();
    {
        TDecoratedFrame::SetupWindow();
        MyChildWindow->ShowWindow(SW_NORMAL);
        MyChildWindow->BringWindowToTop();
    }
};

// Setup a frame window. //

TMyDecoratedFrame::TMyDecoratedFrame(TWindow * parent, const char far * title,
TWindow& clientWnd)
    : TDecoratedFrame(parent, title, clientWnd),
      TFrameWindow(parent, title, &clientWnd),
      TWindow(parent, title)
{
    // Create a new TMyChildWindow. //

    MyChildWindow = new TWindow(this, "");
    MyChildWindow->Attr.Style |= WS_BORDER |WS_VISIBLE |WS_CHILD;
    MyChildWindow->SetBkgndColor( RGB(0,100,0) );

    // Establish metrics for the child window. //

    TLayoutMetrics layoutMetrics;

    layoutMetrics.X.Absolute(lmLeft, 10);
    layoutMetrics.Y.Absolute(lmTop, 10);
    layoutMetrics.Width.Absolute( 80 );
    layoutMetrics.Height.Absolute( 80 );
}

SetChildLayoutMetrics(*MyChildWindow, layoutMetrics);
```

```

class TMyApp : public TApplication {
public:
    virtual void InitMainWindow()
    {
        TWindow* client = new TWindow(0, "title");
        MainWindow = new TMyDecoratedFrame(0, "Layout Window Ex", *client);
    }
};

int OwlMain(int, char**) {
    return TMyApp.Run();
}

```

TLayoutWindow class

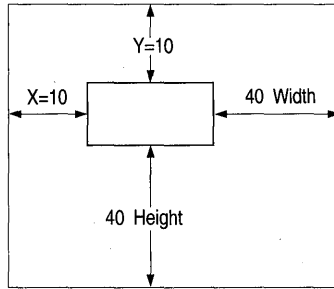
layoutwi.h

TLayoutWindow is derived from *TWindow* and provides options for defining the layout metrics for a window. See *TLayoutConstraint* for a definition of the layout constraints and *TLayoutMetrics* for a description of the metrics you can use to set up layout constraints.

The following examples show how to set up various metrics using edge constraints. For purposes of illustration, these examples use a parent-child relationship, but you can also use a child-to-child (sibling) relationship. Keep in mind that, as usual, if you move the parent's origin (the left and top edges), the child will move with the parent window.

Examples

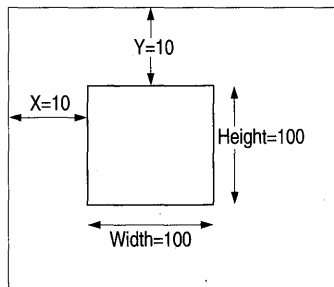
- Example 1** To create growable windows, set the top and left edges of the child window's boundaries in a fixed relationship to the top and left edges of the parent's window. In this example, if you expand the bottom and right edges of the parent, the child's bottom and right edges grow the same amount. Both the X and Y constraints are 10 units from the parent window's edges. Both the Width and Height constraints are 40 layout units from the parent window's edges. Specifically, Width (*lmWidth*) is 40 units to the left of the parent's right edge (*lmLeftOf* = *lmSameAs* + offset or *sameas* - 40).



Use the following layout constraints:

```
layoutmetrics.X.Set(lmLeft, lmRightOf, lmParent, lmLeft, 10);
layoutmetrics.Y.Set(lmTop, lmBelow, lmParent, lmTop, 10);
layoutmetrics.Width.Set(lmRight, lmLeftOf, lmParent, lmRight, 40);
layoutmetrics.Height.Set(lmBottom, lmAbove, lmParent, lmBottom, 40);
SetChildLayoutMetrics(*MyChildWindow, layoutMetrics);
```

Example 2 To create fixed-size and fixed-position windows, set the child's right edge a fixed distance from parent's left edge and the child's bottom edge a fixed distance from the parent's top edge. In this example, both the X and Y edge constraints are set to 10 and both the Width and Height edge constraints are set to 100.

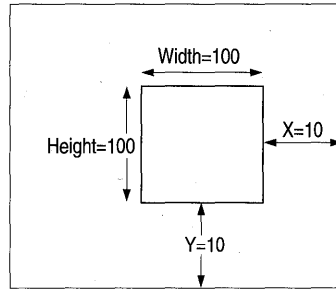


Use the following layout constraints:

```
layoutmetrics.X.Set(lmLeft, lmRightOf, lmParent, lmLeft, 10);
layoutmetrics.Y.Set(lmTop, lmBelow, lmParent, lmTop, 10);
layoutmetrics.Width.Absolute(100);
layoutmetrics.Height.Absolute(100);
SetChildLayoutMetrics(*MyChildWindow, layoutMetrics);
```

Example 3 To create a fixed-size window that remains a constant distance from the parent's right corner, set the child's top and bottom edges a fixed distance (*lmLayout* unit or pixels) from the parent window's bottom. Also, set the child's left and right edges a fixed distance from the parent's right edge. In

this example, both the Width and the Height edge constraints are set to 100 and the X and Y edge constraints are set to 100. In this case, the child window, which stays the same size, moves with the lower right corner of the parent.



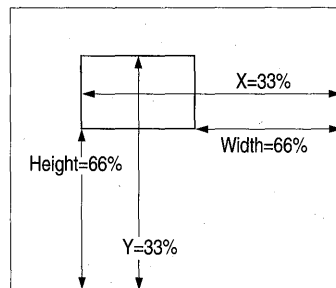
Use the following layout constraints:

```
layoutmetrics.X.Set(lmRight, lmLeftOf, lmParent, lmRight, 10);
layoutmetrics.Y.Set(lmBottom, lmAbove, lmParent, lmBottom, 10);
layoutmetrics.Width.Absolute(100);
layoutmetrics.Height.Absolute(100);
SetChildLayoutMetrics(*MyChildWindow, layoutMetrics);
```

Example 4

To create a window in which the child's edges are a percentage of the parent's window, set the child's edges a percentage of the distance from the parent's edges. Specifically, the child's top and bottom edges are a percentage of the parent's bottom edge. The child's left and right edges are a percentage of the parent's right edge.

If you resize the parent window, the child window will change size and origin (that is, the top and left edges will also change).



Use the following layout constraints:

```
layoutmetrics.X.Set(lmLeft, lmPercentOf, lmParent, lmRight, 33);
layoutmetrics.Y.Set(lmTop, lmPercentOf, lmParent, lmBottom, 33);
```

```
layoutmetrics.Width.Set(lmRight, lmPercentOf, lmParent, lmRight, 66);
layoutmetrics.Height.Set(lmBottom, lmPercentOf, lmParent, lmBottom, 66);
SetChildLayoutMetrics(*MyChildWindow, layoutMetrics);
```

Public constructors and destructor

Constructor

```
TLayoutWindow(TWindow* parent, const char far *title = 0, TModule*
              module = 0);
```

Creates a *TLayoutWindow* object with specified parent, window caption, and library ID.

Destructor

```
~TLayoutWindow();
```

Deletes variables and frees the child metrics and constraints.

Public member functions

GetChildLayoutMetrics `BOOL GetChildLayoutMetrics(TWindow &child, TLayoutMetrics &metrics);`

Gets the layout metrics of the child window.

See also: *TLayoutMetrics::GetChildMetrics*

Layout

```
void Layout();
```

Causes the window to resize and position its children according to the specified metrics. Call *Layout* to implement changes that occur in the layout metrics.

RemoveChildLayoutMetrics `BOOL RemoveChildLayoutMetrics(TWindow &child);`

Removes the layout metrics for a child window.

SetChildLayoutMetrics `void SetChildLayoutMetrics(TWindow &child, TLayoutMetrics &metrics);`

Sets the metrics for the window and removes any existing ones. Set the metrics as shown:

```
layoutMetrics->X.Absolute(lmLeft, 10);
layoutMetrics->Y.Absolute(lmTop, 10);
layoutMetrics->Width.Set(lmWidth, lmRightOf, GetClientWindow(), lmWidth, -40);
layoutMetrics->Height.Set(lmHeight, lmRightOf, GetClientWindow(), lmHeight,
-40);
```

Then call *SetChildLayoutMetrics* to associate them with the position of the child window:

```
SetChildLayoutMetrics(* MyChildWindow, * layoutMetrics);
```


Protected data members

ClientSize

TSize ClientSize;

Contains the size of the client area.

Protected member functions

EvSize

void EvSize(UINT sizeType, TSize& size);

Responds to a change in window size by calling *Layout* to resize the window.

Response table entries

Response table entry	Member function
EV_WM_SIZE	EvSize

TListBox class

listbox.h

A *TListBox* is an interface object that represents a corresponding list box element in Windows. A *TListBox* must be used to create a list box control in a parent *TWindow*. A *TListBox* can be used to facilitate communication between your application and the list box controls of a *TDialog*. *TListBox*'s member functions also serve instances of its derived class, *TComboBox*. *TListBox* is a streamable class.

Public constructors

Constructor

TListBox(TWindow* parent, int Id, int x, int y, int w, int h, TModule* module = 0);

Constructs a list box object with the supplied parent window (*parent*) library ID (*module*), position (*x*, *y*) relative to the origin of the parent window's client area, width (*w*), and height (*h*). Invokes a *TControl* constructor. Adds LBS_STANDARD to the default styles for the list box to provide it with

- A border (WS_BORDER)
- A vertical scroll bar (WS_VSCROLL)

- Automatic alphabetic sorting of list items (LBS_SORT)
- Parent window notification upon selection (LBS_NOTIFY)

The *TListBox* member functions that are described as being for single-selection list boxes are inherited by *TComboBox* and can also be used by combo boxes. Also, these member functions return -1 for multiple-selection list boxes.

See also: *GetSelIndex*, *GetSelString*, *SetSelIndex*, *SetSelString*

Constructor

```
TListBox(TWindow* parent, int resId, TModule* module = 0)
```

Constructs a *TListBox* object to be associated with a list box control of a *TDialog*. Invokes the *TControl* constructor with similar parameters. The *module* parameter must correspond to a list box resource that you define.

See also: *TControl::TControl*

Public member functions

AddString

```
virtual int AddString(const char far* str);
```

Adds *string* to the list box, returning its position in the list (0 is the first position). Returns a negative value if an error occurs. The list items are automatically sorted unless the style LBS_SORT is removed from the list box object's *Attr.Style* data member before creation.

See also: *TListBox::DeleteString*, *TListBox::InsertString*

ClearList

```
inline virtual void ClearList();
```

Clears all items in the list.

DeleteString

```
virtual int inline DeleteString(int index);
```

Deletes the item in the list at the position (starting at 0) supplied in *index*. *DeleteString* returns the number of remaining list items, or a negative value if an error occurs.

See also: *TListBox::AddString*, *TListBox::InsertString*

DirectoryList

```
virtual int inline DirectoryList(UINT attrs, const char far* fileSpec)
```

Adds a list of file names to a list box.

FindExactString

```
int FindExactString(const char far* str, int searchIndex) const;
```

Starting at the line number passed in *searchIndex*, searches the list box for an exact match with the string *str*. If a match is not found after the last string has been compared, the search continues from the beginning of the list until a match has been found or until the list has been completely traversed.

Searches from the beginning of the list when `-1` is supplied as *searchIndex*. Returns the index of the first string found if successful, a negative value if an error occurs.

See also: *TListBox::AddString*, *TListBox::DeleteString*

FindString

```
virtual int inline FindString(const char far* str, int Index) const;
```

Searches the list box as described under *FindExactString*, but looks for the first entry that begins with *str*.

See also: *TListBox::AddString*, *TListBox::DeleteString*, *TListBox::InsertString*

GetCaretIndex

```
int GetCaretIndex() const;
```

Returns the index of the currently focused list-box item. For single-selection list boxes, the return value is the index of the selected item, if one is selected.

See also: *TListBox::SetCaretIndex*

GetCount

```
inline virtual int GetCount() const;
```

Returns the number of items in the list box, or a negative value if an error occurs.

GetHorizontalExtent

```
inline int GetHorizontalExtent() const;
```

Returns the number of pixels by which the list box can be scrolled horizontally.

See also: *TListBox::SetHorizontalExtent*

GetItemData

```
inline virtual DWORD GetItemData(int index) const;
```

Returns the 32-bit value of the list box item set by *SetItemData*.

See also: *TListBox::SetItemData*

GetItemHeight

```
inline virtual int GetItemHeight(int index) const;
```

Returns the height in pixels of the specified list box items.

See also: *TListBox::SetItemHeight*

GetItemRect

```
inline int GetItemRect(int index, TRect& rect) const;
```

Returns the dimensions of the rectangle that surrounds a list-box item currently displayed in the list-box window.

GetSel

```
BOOL GetSel(int index) const;
```

Returns the index of the selected item in the list box.

See also: *TListBox::SetSel*

- GetSelCount** `int GetSelCount() const;`
Returns the number of selected items in the single- or multiple-selection list box or combo box.
- GetSelIndex** `virtual int GetSelIndex() const;`
For single-selection list boxes. Returns the nonnegative index (starting at 0) of the currently selected item, or a negative value if no item is selected.
See also: *TListBox::SetSelIndex*
- GetSelIndexes** `int GetSelIndexes(int* indexes, int maxCount) const;`
For multiple-selection list boxes. Fills the *indexes* array with the indexes of up to *maxCount* selected strings. Returns the number of items put in *indexes* (-1 for single-selection list boxes and combo boxes).
See also: *TListBox::SetSelIndexes*
- GetSelString** `int GetSelString(char far* str, int maxChars) const;`
Retrieves the currently selected items, putting up to *maxChars* of them in *Strings*. Each entry in the *Strings* array should have space for *maxChars* characters and a terminating null. For single-selection list boxes, returns the string length, a negative value if an error occurs, or 1 if no string is selected. For multiple-selection list boxes, returns -1.
See also: *TListBox::SetSelString*
- GetSelStrings** `int GetSelStrings(char far** str, int maxCount, int maxChars) const;`
Retrieves the total number of selected items for a multiselection list and copies them into the buffer. *str* is an array of pointers to chars. Each of the pointers to the buffers is of *maxChars*. *maxCount* is the size of the array.
See also: *TListBox::SetSelStrings*
- GetString** `inline virtual int GetString(char far* str, int index) const;`
Retrieves the item at the position (starting at 0) supplied in *index* and returns it in *str*. *GetString* returns the string length, or a negative value if an error occurs.
- GetStringLen** `inline virtual int GetStringLen(int Index) const;`
Returns the string length (excluding the terminating NULL) of the item at the position *index* supplied in *Index*. Returns a negative value in the case of an error.
- GetTopIndex** `inline int GetTopIndex() const;`

Returns the index of the first item displayed at the top of the list box.

See also: *TListBox::SetTopIndex*

InsertString

```
virtual int InsertString(const char far* str, int index);
```

Inserts *str* in the list box at the position supplied in *index*, and returns the item's actual position (starting at 0) in the list. A negative value is returned if an error occurs. The list is not resorted. If *index* is -1, the string is appended to the end of the list.

See also: *TListBox::AddString*, *TListBox::DeleteString*, *TListBox::FindString*

SetCaretIndex

```
int SetCaretIndex(int index, BOOL partScrollOk);
```

Sets the focus to the item specified at *index*. An item that is not visible is scrolled into view.

See also: *TListBox::GetCaretIndex*

SetColumnWidth

```
inline void SetColumnWidth(int width);
```

Sets the width in pixels of the items in the list box.

SetHorizontalExtent

```
void SetHorizontalExtent(int horzExtent);
```

Sets the number of pixels by which the list box can be scrolled horizontally.

See also: *TListBox::GetHorizontalExtent*

SetItemData

```
inline virtual int SetItemData(int index, DWORD itemData);
```

Sets the 32-bit value of the list box item at the specified *index* position.

See also: *TListBox::GetItemData*

SetItemHeight

```
inline virtual int SetItemHeight(int index, int height);
```

Sets the height in pixels of the items in the list box.

See also: *TListBox::GetItemHeight*

SetSel

```
inline int SetSel(int index, BOOL select);
```

Selects an item at the position specified in *index*. For multiple-selection list boxes.

See also: *TListBox::GetSel*

SetSelIndex

```
virtual int SetSelIndex(int index);
```

For single-selection list boxes. Forces the selection of the item at the position (starting at 0) supplied in *index*. If *index* is -1, the list box is cleared of any selection. *SetSelIndex* returns a negative number if an error occurs.

SetSelIndexes

```
int SetSelIndexes(int* indexes, int numSelections, BOOL shouldSet);
```

For multiple-selection list boxes. Selects/deselects the strings in the associated list box at the indexes specified in the *Indexes* array. If *ShouldSet* is TRUE, the indexed strings are selected and highlighted; if *ShouldSet* is FALSE the highlight is removed and they are no longer selected. Returns the number of strings successfully selected or deselected (-1 for single-selection list boxes and combo boxes). If *NumSelections* is less than 0, all strings are selected or deselected, and a negative value is returned on failure.

SetSelItemRange inline int SetSelItemRange(BOOL select, int first, int last);

Selects the range of items specified from *first* to *last*.

SetSelString int SetSelString(const char far* str, int searchIndex);

For single-selection list boxes. Forces the selection of the first item beginning with the text supplied in *str* that appears beyond the position (starting at 0) supplied in *SearchIndex*. If *SearchIndex* is -1, the entire list is searched, beginning with the first item. *SetSelString* returns the position of the newly selected item, or a negative value in the case of an error.

SetSelStrings int SetSelStrings(const char far** prefixes, int numSelections, BOOL shouldSet);

For multiple-selection list boxes. Selects the strings in the associated list box that begin with the prefixes specified in the *prefixes* array. For each string the search begins at the beginning of the list and continues until a match is found or until the list has been completely traversed. If *shouldSet* is TRUE, the matched strings are selected and highlighted; if *shouldSet* is FALSE the highlight is removed from the matched strings and they are no longer selected. Returns the number of strings successfully selected or deselected (-1 for single-selection list boxes and combo boxes). If *numSelections* is less than 0, all strings are selected or deselected, and a negative value is returned on failure.

SetTabStops inline BOOL SetTabStops(int numTabs, int far* tabs);

numTabs is the number of tabstops. *tabs* is the array of integers representing the tab positions.

SetTopIndex inline int SetTopIndex(int index);

Sets *index* to the first item displayed at the top of the list box.

See also: *TListBox::GetTopIndex*

Transfer virtual WORD Transfer(void *buffer, TTransferDirection direction);

Transfers the items and selection(s) of the list box to or from a transfer buffer if `tdSetData` or `tdGetData`, respectively, is passed as the *direction*. *buffer* is expected to point to a pointer to a *TListBoxData* structure.

Transfer, which overrides the *TWindow* virtual member function, returns the size of *TListBoxData* (the pointer, not the structure). To retrieve the size without transferring data, pass `tdSizeData` as the *direction*.



You must use a pointer in your transfer buffer to these structures. You can't embed copies of the structures in your transfer buffer, and you can't use these structures as transfer buffers.

See also: *TListBoxData*, *TWindow::Transfer*

Protected member functions

GetClassName

```
char far* GetClassName();
```

Returns the name of *TListBox*'s Windows registration class, "LISTBOX."

TListBoxData struct

listbox.h

A *TListBoxData* is a structure that is used to transfer the contents of a list box.

Public data members

ItemDatas

```
TWordArray* ItemDatas;
```

Contains all item data `DWORD` for each item in the list box.

SelCount

```
int SelCount;
```

Holds the number of selected items.

SelIndices

```
TIntArray* SelIndices
```

Contains the indexes of all the selected strings in a multiple-selection list box.

SelStrings

```
TStringArray* SelStrings;
```

Pointer to an array of the strings to select when data is transferred into the list box. When data is transferred out of the list box, *SelStrings* returns the current selection(s).

Strings

```
TStringArray* Strings;
```

Pointer to an array of strings to be transferred into a *TListBox*.

Public constructors and destructor

Constructor

```
TListBoxData();
```

Constructs *Strings* and *SelStrings*. Initializes *SelCount* to 0.

Destructor

```
~TListBoxData();
```

Deletes the space allocated for *Strings* and *SelStrings*.

Public member functions

AddString

```
void AddString(const char *str, BOOL isSelected = FALSE);
```

Adds the specified string to *Strings*. If *IsSelected* is TRUE, adds the string to *SelStrings* and increments *SelCount*.

AddStringItem

```
void AddStringItem(const char* str, DWORD itemData,
                  BOOL isSelected = FALSE);
```

Adds a string to the *Strings* array, optionally selects it, and adds item data to the *ItemDatas* array.

GetSelString

```
void GetSelString(char far* buffer, int bufferSize, int index = 0) const;
```

Locates the string at the specified *index* in *SelStrings* and copies it into *buffer*. *bufferSize* includes the terminating NULL.

GetSelStringLength

```
int GetSelStringLength(int index = 0) const;
```

Returns the length (excluding the terminating NULL) of the string at the specified *index* in *SelStrings*.

ResetSelections

```
void ResetSelections();
```

Removes all strings from *SelStrings* and sets *SelCount* to 0.

Select

```
void Select(int index);
```

Selects the string at the given index.

SelectString

```
void SelectString(const char far* str);
```

Adds *str* to *SelStrings* and increments *SelCount*.

TListView class

listview.h

Derived from *TListBox* and *TView*, *TListView* provides views for list boxes. See *TView* for a description of view functions and *TListBox* for listbox functions.

Public constructors and destructor**Constructor**

```
TListView(TDocument& doc, TWindow* parent = 0);
```

Creates a *TListView* object associated with the specified document and parent window. Sets *Attr.AccelTable* to *IDA_LISTVIEW* to identify the edit view. Sets *Attr.Style* to *WS_HSCROLL | LBS_NOINTEGRALHEIGHT*. Sets *TView::ViewMenu* to the new *TMenuDescr* for this view.

Destructor

```
~TListView();
```

After checking to see if there is an open view, this destructor destroys the *TListView* object.

Public data member**DirtyFlag**

```
BOOL DirtyFlag;
```

Is nonzero if the data in the list view has been changed; otherwise, is 0.

Public member functions**CanClose**

```
inline BOOL CanClose();
```

Checks to see if all child views can be closed before closing the current view. If any child returns 0, *CanClose* returns 0 and aborts the process. If all children return nonzero, it calls *TDocManager::FlushDoc*.

See also: *TView::CanClose*, *TDocManager::FlushDoc*

Create

```
virtual BOOL Create();
```

Overrides *TWindow::Create* and calls *TEditSearch::Create* to create the view's window. Calls *GetDocPath* to determine if the file is new or already has data. If there is data, calls *LoadData* to add the data to the view. If the view's window can't be created, *Create* throws a *TXInvalidWindow* exception.

GetViewName

```
inline LPCSTR GetViewName();
```

Overrides *TView's* virtual function and returns the descriptive name of the class (*StaticName*).

See also: *TView::GetViewName*

GetWindow

```
inline TWindow* GetWindow();
```

Overrides *TView's* virtual function and returns the list view object as a *TWindow*.

See also: *TView::GetWindow*

SetDocTitle

```
inline BOOL SetDocTitle(LPCSTR docname, int index);
```

Overrides *TView's* virtual function and stores the document title. This name is forwarded up the parent chain until a *TFrameWindow* object accepts the data and displays it in its caption.

See also: *TView::SetDocTitle*

StaticName

```
inline static LPCSTR StaticName();
```

Overrides *TView's* function and returns a constant string, "ListView." This information is displayed in the user interface selection box.

See also: *TView::GetViewName*

Protected data members

MaxWidth

```
int MaxWidth;
```

Holds the maximum horizontal extent (the number of pixels by which the view can be scrolled horizontally).

Origin

```
long Origin;
```

Holds the file position at the beginning of the display.

Protected member functions

CmEditAdd

```
void CmEditAdd();
```

Automatically responds to *CM_LISTADD* message by getting the length of the input string and calling *InsertString* to insert the text string into the list view. Sets the data member *DirtyFlag* to TRUE.

CmEditCopy

```
void CmEditCopy();
```

Automatically responds to a menu selection with a menu ID of `CM_EDITCOPY` by calling `TListBox::Copy` to copy the selected text to the Clipboard.

CmEditClear `void CmEditClear();`

Automatically responds to a menu selection with a menu ID of `CM_EDITCLEAR` by calling `TListBox::Clear` to clear the list view.

CmEditCut `void CmEditCut();`

Automatically responds to a menu selection with a menu ID of `CM_EDITCUT` by calling `CmEditCopy` and `CmEditDelete` to delete a text string from the list view. Sets the data member `DirtyFlag` to `TRUE`.

CmEditDelete `void CmEditDelete();`

Automatically responds to a menu selection with a menu ID of `CM_EDITDELETE` by calling `TListBox::DeleteSelection`.

CmEditItem `void CmEditItem();`

Automatically responds to a `CM_LISTEDIT` message by getting the input text and inserting into the list view. Sets the `DirtyFlag` to nonzero to indicate that the view has been changed and not saved.

CmEditPaste `void CmEditPaste();`

Automatically responds to a menu selection with a menu ID of `CM_EDITPASTE` by calling `TListBox::Paste`.

CmEditUndo `void CmEditUndo();`

Automatically responds to a menu selection with a menu ID of `CM_EDITUNDO` by calling `TListBox::Undo`.

CmSelChange `void CmSelChange();`

Automatically responds to a `LBN_SELCHANGE` message (which indicates that the contents of the list view have changed) by calling `DefaultProcessing`.

EvGetDlgCode `UINT EvGetDlgCode();`

Overrides `TWindow's` response to a `WM_GETDLGCODE` message (an input procedure associated with a control that isn't a check box) by calling `DefaultProcessing`.

LoadData `BOOL LoadData(int top, int sel);`

Reads the view from the stream and closes the file. Returns `TRUE` if the view was successfully loaded.

Throws an *xmsg* exception and displays the error message “TListView initial read error” if the file can’t be read. Returns FALSE if the view can’t be loaded.

SetExtent void SetExtent(LPSTR str);

Sets the maximum horizontal extent for the list view window.

VnCommit BOOL VnCommit(BOOL force);

VnCommit commits changes made in the view to the document. If *force* is nonzero, all data, even if it’s unchanged, is saved to the document.

See also: *TListView::vnRevert*, *vnxxxx* view notification constants

VnDocClosed BOOL VnDocClosed(int omode);

VnDocClosed indicates that the document has been closed.

See also: *vnxxxx* view notification constants

VnIsDirty inline BOOL VnIsDirty();

VnIsDirty returns nonzero if changes made to the data in the view have not been saved to the document; otherwise, returns 0.

See also: *vnxxxx* view notification constants

VnIsWindow inline BOOL VnIsWindow(HWND hWnd);

VnIsWindow returns nonzero if the window’s handle passed in *hWnd* is the same as that of the view’s display window.

See also: *vnxxxx* view notification constants

VnRevert BOOL VnRevert(BOOL clear);

VnRevert indicates if changes made to the view should be erased, and the data from the document should be restored to the view. If *clear* is nonzero, the data is cleared instead of restored to the view.

See also: *TListView::vnCommit*

Response table entry

Entry	Member function
EV_COMMAND(CM_LISTUNDO, CmEditUndo)	CmEditUndo
EV_COMMAND(CM_LISTCUT, CmEditCut)	CmEditcut
EV_COMMAND(CM_LISTCOPY, CmEditCopy)	CmEditCopy
EV_COMMAND(CM_LISTPASTE, CmEditPaste)	CmEditPaste
EV_COMMAND(CM_LISTCLEAR, CmEditClear)	CmEditClear

Entry	Member function
EV_COMMAND(CM_LISTDELETE, CmEditDelete)	CmEditDelete
EV_COMMAND(CM_LISTADD, CmEditAdd)	CmEditAdd
EV_COMMAND(CM_LISTEDIT, CmEditItem)	CmEditItem
EV_WM_GETDLGCODE	EvGetDlgCode
EV_NOTIFY_AT_CHILD(LBN_DBLCLK, CmEditItem)	CmEditItem
EV_NOTIFY_AT_CHILD(LBN_SELCHANGE, CmSelChange)	CmSelChange
EV_VN_DOCCLOSED	VnDocClosed
EV_VN_ISWINDOW	VnIsWindow
EV_VN_ISDIRTY	VnIsDirty
EV_VN_COMMIT	VnCommit
EV_VN_REVERT	VnRevert

TLookupValidator class

validate.h

A streamable class, *TLookupValidator* compares the string typed by a user with a list of acceptable values. *TLookupValidator* is an abstract validator type from which you can derive useful lookup validators. You will never create an instance of *TLookupValidator*. When you create a lookup validator type, you need to specify a list of valid items and override the *Lookup* method to return TRUE only if the user input matches an item in that list. One example of a working descendant of *TLookupValidator* is *TStringLookupValidator*.

Public constructors

Constructor

```
TLookupValidator();
```

Constructs a *TLookupValidator* object.

Public member functions

IsValid

```
BOOL IsValid(const char far* str);
```

IsValid overrides *TValidator*'s virtual function and calls *Lookup* to find the string *str* in the list of valid input items. *IsValid* returns TRUE if *Lookup* returns TRUE, meaning *Lookup* found *str* in its list; otherwise, it returns FALSE.

Lookup

```
virtual BOOL Lookup(const char far* str);
```

Searches for the string *str* in the list of valid entries and returns TRUE if it finds *str*; otherwise, returns FALSE. *TLookupValidator*'s *Lookup* is an abstract

method that always returns FALSE. Descendant lookup validator types must override *Lookup* to perform a search based on the actual list of acceptable items.

TMDIChild class

mdichild.h

TMDIChild defines the basic behavior of all MDI child windows. To be used as MDI children, classes must be derived from *TMDIChild*. MDI children can inherit keyboard navigation, focus handling, and icon support from *TFrameWindow*. *TMDIChild* is a streamable class.

Public constructors and destructor

Constructor

```
TMDIChild(TMDIClient &parent, const char far *title = 0,
          TWindow *clientWnd = 0, BOOL shrinkToClient = FALSE, TModule*
          module = 0);
```

Creates an MDI child window of the MDI client window specified by *parent*, using the specified *title*, client window (*clientWnd*) and instance (*inst*). Invokes the *TFrameWindow* base class constructor, supplying *parent*, *title*, *clientWnd*, *inst*, and indicating that the child window is not to be resized to fit. Invokes the *TWindow* base class constructor, specifying *parent*, *title*, and *inst*. The window attributes are then adjusted to include `WS_VISIBLE`, `WS_CHILD`, `WS_CLIPSIBLINGS`, `WS_CLIPCHILDREN`, `WS_SYSMENU`, `WS_CAPTION`, `WS_THICKFRAME`, `WS_MINIMIZEBOX`, and `WS_MAXIMIZEBOX`. The dimensions of the window are set to the Windows default values.

Constructor

```
TMDIChild(HWND hWnd, TModule* module = 0);
```

Creates an MDI child window object from a preexisting window, specified by *hWnd*. The base class *TFrameWindow* constructor is invoked, specifying this *hWnd*, as well as the specified *inst*. The base class *TWindow* constructor is invoked, supplying the *hWnd* and *inst* parameters.

Destructor

```
~TMDIChild();
```

Destructs the MDI child window object.

Public member functions

Destroy

```
void Destroy(int = 0);
```

Destroys the interface element associated with the *TMDIChild*. Calls *EnableAutoCreate* for each window in the child list so that the children are also re-created when the parent window is re-created.

See also: *TWindow::EnableAutoCreate*

PreProcessMsg

BOOL PreProcessMsg (MSG &);

Performs preprocessing of window messages for the MDI child window. If keyboard handling is enabled the parent client window's *TMDIClient::PreProcessMsg* member function is called to preprocess messages. In this case, the return value is TRUE. Otherwise, *TFrameWindow::PreProcessMsg* is called and its return value becomes the return value of this member function.

See also: *TMDIClient::PreProcessMsg*, *TFrameWindow::PreprocessMsg*

Protected member functions

DefWindowProc

LRESULT DefWindowProc (UINT msg, WPARAM wParam, LPARAM lParam);

Overrides *TWindow::DefWindowProc* and calls the Windows API function *::DefMDIChildProc* that provides default processing for any incoming message the MDI child window does not process.

See also: *::DefMDIChildProc*, *TWindow::DefWindowProc*

EvMDIActivate

void EvMDIActivate(BOOL activate, HWND hWndActivated,
 HWND hWndDeactivated);

Instructs a client window to activate or deactivate an MDI child window and then sends a message to the child window being activated and the child window being deactivated.

PerformCreate

void PerformCreate(int menuOrId);

Creates the interface element associated with the MDI child window. If no parent window has been specified, *PerformCreate* throws a *TWindow::TXInvalidChildWindow* exception. Otherwise, it calls *::SendMessage* to notify the parent MDI client window to create the child window's interface element. The supplied *menuOrId* parameter is ignored because MDI child windows cannot have menus.

See also: Windows message WM_MDICREATE, *::SendMessage*

Response table entries

Response table entry	Member function
EV_WM_MDIACTIVATE	EvMDIActivate

TMDIClient class

mdi.h

Multiple Document Interface (MDI) client windows (represented by a *TMDIClient* object) manage the MDI child windows of a *TMDIFrame* parent. *TMDIClient* is a streamable class.

Public data member

ClientAttr

```
LPCLIENTCREATESTRUCT ClientAttr;
```

ClientAttr holds a pointer to a structure of the MDI client window's attributes.

Public constructors and destructor

Constructor

```
TMDIClient(TModule* module = 0);
```

Creates an MDI client window object by invoking the base class *TWindow* constructor, passing it a null parent window, a null title, and the specified library ID. Sets *Attr.Id* to the default client window identifier (IDW_MDICLIENT) and sets *Attr.Style* to include MDIS_ALLCHILDSTYLES, WS_GROUP, WS_TABSTOP, WS_CLIPCHILDREN, WS_VSCROLL, and WS_HSCROLL. Initializes the *ClientAttr* data member, setting its *idFirstChild* member to IDW_FIRSTMDICHILD.

See also: *TWindow::TWindow*

Destructor

```
~TMDIClient();
```

Frees the *ClientAttr* structure.

See also: *TWindow::~~TWindow*

Public member functions

ArrangeIcons

```
inline virtual void ArrangeIcons();
```


Arranges the MDI child window icons at the bottom of the MDI client window.

CascadeChildren inline virtual void CascadeChildren();

Sizes and arranges all of the non-iconized MDI child windows within the MDI client window. The children are overlapped, although each title bar is visible.

CloseChildren virtual BOOL CloseChildren();

First calls *CanClose* on each of the MDI child windows owned by this MDI client. Returns TRUE if all MDI children are closed; otherwise returns FALSE.

See also: *TWindow::CanClose*

Create BOOL Create();

Creates the interface element associated with the MDI client window. Calls *TWindow::Create* after first setting the child window menu in *ClientAttr* to the parent frame window's child menu.

See also: *TWindow::Create*, *TFrameWindow::ChildMenuPos*, *TFrameWindow::GetMenu*

CreateChild virtual TWindow* CreateChild();

Overrides member function defined by *TWindow*. Constructs and creates a new MDI child window by calling *InitChild* and *Create*. Returns a pointer to the new MDI child window.

See also: *TMDIClient::InitChild*, *TModule::MakeWindow*, *TWindow::Create*

GetActiveMDIChild TMDIChild *GetActiveMDIChild();

GetActiveMDIChild points to the *TMDIClient*'s active MDI child window. *GetActiveMDIChild* is set by the child in its *EvMDIActivate* message response member function. *TMDIClient*'s constructors initialize *GetActiveChild*.

InitChild inline virtual TMDIChild *InitChild();

Constructs an instance of *TWindow* as an MDI child window and returns a pointer to it. Children must be created with MDI client as the parent window. Redefine this member function in your derived MDI window class to construct an instance of a derived MDI child class. For example,

```
PTWindowsObject TMyMDIClient::InitChild()
{
    return new TMyMDIChild(this, "");
}
```

See also: *TMDIClient::CreateChild*

PreProcessMsg

```
BOOL PreProcessMsg(MSG &msg);
```

If the specified *msg* is one of `WM_KEYDOWN` or `WM_SYSKEYDOWN`, then the Windows *TranslateMDISysAccel* function is called to translate keyboard accelerators for the MDI client.

See also: *TWindow::PreProcessMsg*

TileChildren

```
inline virtual void TileChildren(int tile = MDITILE_VERTICAL);
```

Sizes and arranges all of the non-iconized MDI child windows within the MDI client window. The children fill up the entire client area without overlapping.

Protected member functions

CmArrangeIcons

```
inline void CmArrangeIcons();
```

Calls *ArrangeIcons* in response to a menu selection with an ID of `CM_ARRANGEICONS`.

See also: *TMDIClient::ArrangeIcons*

CmCascadeChildren

```
inline void CmCascadeChildren();
```

Calls *CascadeChildren* in response to a menu selection with an ID of `CM_CASCADECHILDREN`.

See also: *TMDIClient::CascadeChildren*

CmChildActionEnable

```
void CmChildActionEnable(TCommandEnabler& commandEnabler);
```

If there are MDI child windows, *CmChildActionEnable* enables any one of the child window action menu items.

CmCloseChildren

```
inline void CmCloseChildren();
```

Calls *CloseChildren* in response to a menu selection with an ID of `CM_CLOSECHILDREN`.

See also: *TMDIClient::CloseChildren*

CmCreateChild

```
inline void CmCreateChild();
```

Calls *CreateChild* to produce a new child window in response to a menu selection with a menu ID of `CM_CREATECHILD`.

See also: *TMDIClient::CreateChild*

CmTileChildren

```
inline void CmTileChildren();
```

Calls *TileChildren* in response to a menu selection with an ID of `CM_TILECHILDREN`.

See also: *TMDIClient::TileChildren*

CmTileChildrenHoriz inline void CmTileChildrenHoriz();

Calls *TileChildren* in response to a menu selection with an ID of `CM_TILECHILDREN` and passes MDI child title flag as `MDITILE_HORIZONTAL`.

EvMDICreate

LRESULT EvMDICreate(MDICREATESTRUCT far& createStruct);

Intercepts the `WM_MDICREATE` message sent when MDI child windows are created, and, if the client's style includes `MDIS_ALLCHILDSTYLES`, and the child window's specified style is 0, then changes the child window style attributes to `WS_VISIBLE`, `WS_CHILD`, `WS_CLIPSIBLINGS`, `WS_CLIPCHILDREN`, `WS_SYSMENU`, `WS_CAPTION`, `WS_THICKFRAME`, `WS_MINIMIZEBOX`, and `WS_MAXIMIZEBOX`. Then returns the value of calling *DefaultProcessing*.

See also: *TWindow::DefaultProcessing*

GetClassName

char far *GetClassName();

Returns *TMDIClient's* Windows registration class name, "MDICLIENT."

Response table entries

Response table entry	Member function
EV_COMMAND (CM_ARRANGEICONS, CmArrangeIcons)	CmArrangeIcons
EV_COMMAND (CM_CASCADECHILDREN, CmCascadeChildren)	CmCascadeChildren
EV_COMMAND (CM_CLOSECHILDREN, CmCloseChildren)	CmCloseChildren
EV_COMMAND (CM_CREATECHILD, CmCreateChild)	CmCreateChild
EV_COMMAND (CM_TILECHILDREN, CmTileChildren)	CmTileChildren
EV_COMMAND (CM_TILECHILDRENHORIZ, CmTileChildrenHoriz)	CmTileChildrenHoriz
EV_COMMAND_ENABLE (CM_TILECHILDREN, CmChildActionEnable)	CmChildActionEnable
EV_COMMAND_ENABLE (CM_CASCADECHILDREN, CmChildActionEnable)	CmChildActionEnable
EV_COMMAND_ENABLE (CM_ARRANGEICONS, CmChildActionEnable)	CmChildActionEnable
EV_COMMAND_ENABLE (CM_CLOSECHILDREN, CmChildActionEnable)	CmChildActionEnable
EV_WM_MDICREATE	EvMDICreate

TMDIFrame class**mdi.h**

Multiple Document Interface (MDI) frame windows, represented by *TMDIFrame*, are overlapped windows that serve as main windows of MDI-compliant applications. *TMDIFrame* objects automatically handle the creation and initialization of an MDI client “window” (represented by a *TMDIClient* object) required by Windows. *TMDIFrame* sets window style *WS_CLIPCHILDREN* by default so that minimal flicker occurs when the MDI frame erases its background and the backgrounds of its children. *TMDIFrame* is a streamable class.

Because *TMDIFrame* is derived from *TFrameWindow*, it inherits keyboard navigation. As a result, all children of the MDI frame acquire keyboard navigation. However, it’s best to enable keyboard navigation only for those children who require it.

Public data members**ChildMenuPos**

```
int ChildMenuPos;
```

ChildMenuPos contains the position in the MDI window’s top-level menu of the child window submenu. The zero position is at top left.

Public constructors**Constructor**

```
TMDIFrame(const char far *title, TResId menuResId,
           TMDIClient &clientWnd = *new TMDIClient, TModule* module = 0);
```

Constructs an MDI frame window object using the caption (*title*) and resource ID (*menuResId*). If no client window is specified (*clientWnd*), then an instance of *TMDIClient* is created automatically and used as the client window of the frame. The supplied library ID (*module*) is passed to the *TFrameWindow* constructor along with a null parent window pointer, caption, client window, and a flag indicating that the client window is not to be resized to fit. The *TWindow* constructor is also invoked; it passes the supplied caption and library ID, as well as a null parent window pointer. Then the child menu position is initialized to be the leftmost menu item, and the supplied menu resource ID is used in a call to *AssignMenu*.

See also: *TMDIClient::TMDIClient*, *TFrameWindow::TFrameWindow*, *TWindow::TWindow*, *TFrameWindow::AssignMenu*

Constructor

```
TMDIFrame(HWND hWindow, HWND clientHwnd, TModule* module = 0);
```

Constructs an MDI frame window using an already created non-ObjectWindows window. Invokes the *TFrameWindow* and *TWindow* constructors, passing in the window handle (*hWindow*) and library ID (*module*). Initializes the child menu position to the leftmost menu item, and constructs a *TMDIClient* object that corresponds to the supplied *clientHWnd*.

See also: *TFrameWindow::TFrameWindow*, *TWindow::TWindow*, *TMDIClient::TMDIClient*

Public member functions

GetClientWindow

```
inline TMDIClient *GetClientWindow ();
```

Returns a pointer to the MDI client window.

See also: *TFrameWindow::GetClientWindow*

SetMenu

```
BOOL SetMenu (HMENU);
```

Looks for the MDI submenu in the new menu bar and updates *ChildMenuPos* if the menu is found. Searches for the MDI child menu in the new menu bar and updates the child menu position (*ChildMenuPos*) with the specified menu index. Then sends the client window an *WM_MDISETMENU* message to set the new menu and invokes *TWindow::DrawMenuBar* to redraw the menu. Returns FALSE if the MDI client indicates that there was no previous menu; otherwise returns TRUE.

See also: *TWindow::DrawMenuBar*

Protected member function

DefWindowProc

```
LRESULT DefWindowProc (UINT message, WPARAM wParam, LPARAM lParam);
```

Overrides *TWindow::DefWindowProc* and calls the Windows API function *::DefFrameProc* that provides default processing for any incoming message the MDI frame window does not process.

See also: *::DefFrameProc*, *::DefWindowProc*

Response table entries

The *TMDIFrame* response table has no entries.

TMeasurementUnits enum**layoutco.h**

```
enum TMeasurementUnits;
```

Used by the *TLayoutConstraint* struct, *TMeasurementUnits* enumerates the measurement units (*lmPixels* or *lmLayoutUnits*) that control the dimension of the window. These can be either pixels or layout units that are obtained by dividing the font height into eight vertical and eight horizontal segments.

See also: *TLayoutConstraint struct*

TMemoryDC class**dc.h**

A DC class derived from *TDC*, *TMemoryDC* provides access to a memory DC.

Public constructors**Constructor**

```
TMemoryDC();
```

Default constructor for a memory DC object.

See also: *TDC::TDC*

Constructor

```
TMemoryDC(const TDC& DC);
```

Creates a memory DC object compatible with the given *DC* argument.

See also: *TDC::TDC*

Public member functions**RestoreBitmap**

```
inline void RestoreBitmap();
```

Restores the originally selected bitmap object for this DC.

See also: *TDC::RestoreObjects*

RestoreObjects

```
inline void RestoreObjects();
```

Restores the originally selected brush, pen, font, palette, and bitmap objects for this DC.

See also: *TDC::RestoreObjects*, *TMemoryDC::RestoreBitmap*

SelectObject

```

inline void SelectObject(const TBrush& brush);
inline void SelectObject(const TPen& pen);
inline void SelectObject(const TFont& font);
inline void SelectObject(const TPalette& palette, BOOL
                        forceBackground=FALSE));
void SelectObject(const TBitmap& bitmap);

```

Selects the given GDI object into this DC.

See also: *TDC::SelectObject*, *TMemoryDC::RestoreObjects*,
TMemoryDC::RestoreBitmap

Protected data member

OrgBitmap

```

HBITMAP OrgBitmap;

```

The original bitmap selected into this DC.

See also: *TMemoryDC::SelectObject*, *TMemoryDC::RestoreBitmap*

TMenu class**menu.h**

The *TMenu* class encapsulates window menus. You can use *TMenu* member functions to construct, modify, query, and create menu objects. You can also use *TMenu* to add bitmaps to your menu or to specify if a menu item is checked or unchecked.

Public constructors and destructor

Constructor

```

TMenu(TAutoDelete autoDelete = AutoDelete);

```

Creates an empty menu and sets *autoDelete*, by default, so that the menu is automatically deleted when the object is destructed.

Constructor

```

TMenu(HWND wnd, TAutoDelete autoDelete = NoAutoDelete);

```

Creates a menu object representing the window's current menu and sets *autoDelete*, by default, so that the menu is not automatically deleted when the object is deconstructed.

Constructor

```

TMenu(HMENU handle, TAutoDelete autoDelete = NoAutoDelete);

```

Creates a menu object from an already loaded menu and sets *autoDelete*, by default, so the menu is not automatically deleted when the object is deconstructed.

- Constructor** `TMenu(LPCVOID* menuTemplate);`
Creates a menu object from a menu template in memory.
- Constructor** `TMenu(HINSTANCE instance, TResId resId);`
Creates a menu object from a specified resource ID.
- Destructor** `virtual ~TMenu();`
Destroys the pop-up menu.

Public member functions

- AppendMenu** `inline BOOL AppendMenu(UINT flags, UINT idNewItem, const TBitmap& newBmp);`
Adds a bitmap menu item at the end of the menu.
- AppendMenu** `inline BOOL AppendMenu(UINT flags, UINT idNewItem = -1,
const char far* newItem = 0);`
Adds a text menu item to the end of the menu.
- CheckMenuItem** `inline BOOL CheckMenuItem(UINT idItem, UINT check);`
Checks or unchecks the menu item.
- DeleteMenu** `inline BOOL DeleteMenu(UINT idItem, UINT flags);`
Removes the menu item (*idItem*) from the menu or deletes the menu item if it's a pop-up menu. *flags* is used to identify the position of the menu item by its relative position in the menu (MF_BYPOSITION) or by referencing the handle to the top-level menu (MF_BYCOMMAND). See the Windows API online Help for information on these flags.
See also: *TMenu::RemoveMenu*
- EnableMenuItem** `inline BOOL EnableMenuItem(UINT idItem, UINT enable);`
Enables or disables the menu options (MF_CHECKED, MF_GRAYED, MF_HELP, MF_MENUBARBREAK, MF_MENUBREAK, MF_OWNERDRAW, MF_POPUP) that control the appearance of the menu item. See the Windows API online Help for information about these flags.
- GetMenuCheckMarkDimensions** `inline static BOOL GetMenuCheckMarkDimensions(TSize& size);`
Gets the size of the bitmap used to display the default check mark on checked menu items.
See also: *TMenu::SetMenuItemBitmaps*
- GetMenuItemCount** `inline UINT GetMenuItemCount() const;`

Returns the number of items in a top-level or pop-up menu.

GetMenuItemID

```
inline UINT GetMenuItemID(int posItem) const;
```

Returns the ID of the menu item at the position specified by *posItem*.

GetMenuState

```
inline UINT GetMenuState(UINT idItem, UINT flags) const;
```

Returns the values of the flags for a menu item specified by *idItem*. *flags* is a combination of one or more of the Windows API *MF_XXXX* flags, such as *MF_POPUP*, *MF_GRAYED*, *MFCHECKED*, and so on.

See also: *::MF_XXXX* flags

GetMenuString

```
inline UINT GetMenuString(UINT idItem, char* str, int count, UINT flags)
    const;
```

Returns the label (*str*) of the menu item (*idItem*).

GetSubMenu

```
inline HMENU GetSubMenu(int posItem) const;
```

Returns the handle of the menu specified by *posItem*.

InsertMenu

```
inline BOOL InsertMenu(UINT idItem, UINT flags, UINT idNewItem,
    const TBitmap& newBmp);
```

Adds a bitmap menu item after the menu item specified in *idItem*.

InsertMenu

```
inline BOOL InsertMenu(UINT idItem, UINT flags, UINT idNewItem = -1,
    const char far* newItem = 0);
```

Inserts a new text menu item or pop-up menu into the menu after the menu item specified in *idItem*.

IsOK

```
BOOL IsOK() const;
```

Returns TRUE if the menu has a valid handle.

ModifyMenu

```
inline BOOL ModifyMenu(UINT idItem, UINT flags, UINT idNewItem,
    const TBitmap& newBmp);
```

Changes an existing menu item into a bitmap.

ModifyMenu

```
inline BOOL ModifyMenu(UINT idItem, UINT flags, UINT idNewItem = -1,
    const char far* newItem = 0);
```

Changes an existing menu item from the item specified in *idItem* to *idNewItem*.

operator HMENU

```
operator HMENU();
```

Returns the menu's handle.

See also: *TMenu::UINT*

- operator UINT** operator UINT();
- Returns the menu's handle. This function provides compatibility with some Windows functions that require a UINT menu parameter.
- See also: *TMenu::HMenu*
- RemoveMenu** inline BOOL RemoveMenu(UINT idItem, UINT flags);
- Removes the menu item from the menu but does not delete it if it is a sub-menu.
- See also: *TMenu::DeleteMenu*
- SetMenuItemBitmaps** inline BOOL SetMenuItemBitmaps(UINT idItem, UINT flags,
 const TBitmap* bmpUnchecked=0,
 const TBitmap* bmpChecked=0);
 static BOOL GetMenuCheckmarkDimensions(TSize& size
- Specifies the bitmap to be displayed when the menu item is checked and unchecked. *idItem* indicates the menu item to be associated with the bitmap. *flags* indicates how the *size* parameter is interpreted (whether by MF_BYPOSITION or by MF_BYCOMMAND). *GetMenuCheckmarkDimensions* gets the size of the bitmap.
- See also: *TMenu::GetMenuCheckmarkDimensions*

Protected data members

- Handle** operator HMENU Handle;
- Holds the handle to the menu.
- ShouldDelete** BOOL ShouldDelete;
- ShouldDelete* is set to TRUE if the destructor needs to delete the handle to the menu.
- See also: *TMenu::DeleteMenu*, *TMenu::RemoveMenu*

TMenuDescr class

framewin.h

TMenuDescr describes your menu bar and its arrangement. It uses a resource ID to identify the menu resource and an array of count values to indicate the number of menus in each group on the menu bar. Actually, *TMenuDescr*'s constructor simply initializes the members based on the

arguments passed: *TFrameWindow's MergeMenu* function performs the real work of merging the menu groups.

For example, if your original menu looked like this:

```
File Edit Search View Page Paragraph Word Window Help
```

you might use the following group counts:

Group	Count	Menu
FileGroup	1	File
EditGroup	2	Edit Search
ContainerGroup	1	View
ObjectGroup	3	Page Paragraph Word
WindowGroup	1	Window
HelpGroup	1	Help

Then invoke the constructor in this way:

```
TMenuDescr (IDM_MYMENU, 1, 2, 1, 3, 1, 1)
```

You can build the previous menu by merging two menus. Set your application's frame menu bar this way:

```
File View Window Help
```

```
TMenuDescr (IDM_FRAME, 1, 0, 1, 0, 1, 1)
```

and the word-processor child menu bar this way:

```
Edit Search Page Paragraph Word Help
```

```
TMenuDescr (IDM_WPROC, 0, 2, 0, 3, 0, 1)
```

If no child is active, only the frame menu will be active. When the word processor child window becomes active, the child menu bar is merged with the frame menu. Every group that is 0 in the child menu bar leaves the parent's group intact. The previous example interleaves every group except for the last group, the Help group, in which the child replaces the frame menu.

By convention, the even groups (File, Container, Window) usually belong to the outside frame or container, and the odd groups (Edit, Object, Help) belong to the child or contained group.

If a -1 is used in a group count, the merger eliminates the parent's group without replacing it. For example, another child menu bar, such as a calculator, could be added to your application in this way.

```
Edit Base Help
```

```
TMenuDescr(IDM_WCALC, 0, 1, -1, 1, 0, 1)
```

This produces a merged menu (with the View menu selection eliminated as a result of the `-1`) that looks like this:

```
File Edit Base Window Help
```

You could add a paint window in this way:

```
Edit Bitmap Pixel Help
```

```
TMenuDescr(IDM_WPAINT, 0, 1, 0, 2, 0, 1)
```

This produces the following merged menu:

```
File Edit View Bitmap Pixel Window Help
```

Public data members

Id

```
TResId Id;
```

Resource ID for the menu.

```
GroupCount[NumGroups] int GroupCount[NumGroups];
```

An array of values indicating the number of pop-up menus within each group on the menu bar.

See also: *TMenuDescr::TGroup* enum

Public Constructors

Constructor

```
TMenuDescr();
```

Default constructor for a *TMenuDescr* object.

Constructor

```
TMenuDescr(TResId id, int fg, int eg, int cg, int og, int wg, int hg);
```

Constructs a *TMenuDescr* object with the specified resource ID and number of items in the file menu group (*fg*), edit menu group (*eg*), container menu group (*cg*), object menu group (*og*), window menu group (*wg*), and help menu group (*hg*).

TMenuDescr:: TGroup enum

framewin.h

enum TGroup

Used by *TMenuDescr*, the *TGroup* enum describes the following constants that define the index of the entry in the *GroupCount* array.

Constant	Meaning
FileGroup	Index of the File menu group count
EditGroup	Index of the Edit menu group count
ContainerGroup	Index of the Container menu group count
ObjectGroup	Index of the Object group count
WindowGroup	Index of the Window menu group count
HelpGroup	Index of the Help menu group count
NumGroups	Total number of groups

See also: *TMenuDescr::GroupCount[NumGroups]*

TMessageBar class

messageb.h

Derived from *TGadgetWindow*, *TMessageBar* implements a message bar with one text gadget as wide as the window and no border. Normally positioned at the bottom of the window, the message bar uses the default gadget window font and draws a highlighted line at the top.

Public constructors**Constructor**

```
TMessageBar(TWindow* parent = 0, TFont* font = new TGadgetWindowFont,
            TModule* module = 0);
```

Constructs a *TMessageBar* object with the gadget window font. Sets *Attr.Id* to *IDW_STATUSBAR*, *HighlightLine* to *TRUE*, and *TTextGadget*'s member *WideAsPossible* to *TRUE*, making the text gadget as wide as the window.

See also: *TGadgetWindowFont::TGadgetWindowFont*

Public member function**SetHintText**

```
virtual void SetHintText(const char* text);
```

Sets or clears the menu hint text for the message bar. Hint text is displayed over all other gadgets and is used for menu and control bar button help.

SetText

```
void SetText(const char* text);
```

Forwards the message in the message bar to the text gadget for formatting.

See also: *TTextGadget::SetText*

Protected data member

Highlightline

BOOL HighlightLine;

Is TRUE if a highlighted line is drawn.

Protected member functions

GetDesiredSize

void GetDesiredSize(TSize& rect);

Calls *TGadgetWindow's GetDesiredSize* to get the size of the message bar. Then, if a highlighting line is drawn, adjusts the size of the message bar.

See also: *TGadgetWindow::GetDesiredSize*.

GetInnerRect

void GetInnerRect(TRect& rect);

GetInnerRect computes the rectangle inside the borders and margins of the message bar.

See also: *TGadgetWindow::GetInnerRect*

PaintGadgets

void PaintGadgets(TDC& dc, BOOL erase, TRect& rect);

Adjusts the message bar and paints a highlight line. Then, *PaintGadgets* either paints the hint text if any is set or calls *TGadgetWindow::PaintGadgets* to repaint each gadget.

See also: *TGadgetWindow::PaintGadgets*

TMetaFileDC class

dc.h

Derived from *TDC*, *TMetaFileDC* provides access to a DC with a metafile selected for drawing.

Constructors and destructor

Constructor

TMetaFileDC(const char far* filename = 0);

Default constructor for *TMetaFileDC* objects.

See also: *TDC::TDC*

Constructor `TMetaFileDC(const char far* filename = 0);`
 Creates a *TMetaFileDC* object with the data in the named file.

See also: *TDC::TDC*

Destructor `~TMetaFileDC();`
 Destroys this object.

Public member function

Close `inline HMETAFILE Close();`
 Closes this metafile DC object. Sets the *Handle* data member to 0 and returns a pointer to a new *TMetaFilePict* object.
 See also: *::CloseMetaFile*, *TMetaFilePict*, *::DeleteMetaFile*

TMetaFilePict class

metafile.h

TMetaFilePict is a support class used with *TMetaFileDC* to simplify Windows-format metafile operations, such as playing into a DC or storing data on the Clipboard.

Protected data member

Extent `TSize Extent;`
 Holds the extent (size) of the metafile.

Public constructors and destructor

Constructor `TMetaFilePict(HMETAFILE handle, TAutoDelete autoDelete);`
 Creates a *TMetaFilePict* object with *Handle* set to the given *handle* argument.
 See also: *TMetaFilePict::Handle*

Constructor `TMetaFilePict(const TClipboard& clipboard);`
 Creates a *TMetaFilePict* object and sets *Handle* from the contents of the specified Clipboard.

See also: *TClipboard*, *::GetClipboardData*, *TMetaFilePict::Handle*

Constructor

```
TMetaFilePict(const char* filename);
```

Creates a *TMetaFilePict* object for the Windows-format metafile stored in the named file by calling *::GetMetaFile(filename)*. *Handle* is set from the return value of the *::GetMetaFile(filename)* call.

See also: *::GetMetaFile*, *TMetaFilePict::Handle*

Constructor

```
TMetaFilePict(UINT size, const void far* data);
```



Creates a *TMetaFilePict* object for the memory-based metafile specified by *data* by calling *::SetMetaFileBitsEx(size, data)*. The *data* buffer must hold a Windows-format metafile of length *size* bytes. *Handle* is set from the return value of the *::SetMetaFileBitsEx* call.

See also: *::SetMetaFileBitsEx*, *TMetaFilePict::GetMetaFileBitsEx*, *TMetaFilePict::Handle*

Constructor

```
TMetaFilePict(HGLOBAL data);
```



Creates a *TMetaFilePict* object for the memory-based metafile specified by *data* by calling *::SetMetaFileBitsBetter(data)*. The *data* global memory block must hold a Windows-format metafile. *Handle* is set from the return value of the *::SetMetaFileBitsBetter* call.

See also: *::SetMetaFileBitsBetter*, *TMetaFilePict::GetMetaFileBits*, *TMetaFilePict::Handle*

Constructor

```
TMetaFilePict(const TMetaFilePict& orig, const char far* fileName = 0);
```

Copies the Windows-format metafile, *orig*, to the named file by calling *::CopyMetaFile(filename)*. If *filename* is 0 (the default), the metafile is copied to a memory-based metafile. *Handle* is set from the return value of the *::CopyMetaFile* call.

See also: *::CopyMetaFile*, *TMetaFilePict::Handle*

Destructor

```
~TMetaFilePict()
```

Destroys this object. If *Handle* is nonzero, the associated metafile is also destroyed via *::DeleteMetaFile*.

See also: *::DeleteMetaFile*, *TMetaFilePict::Handle*

Public member functions

GetMetaFileBits



```
inline HANDLE GetMetaFileBits();
```

Returns a handle to a global memory block containing this metafile as a collection of bits by calling `::GetMetaFileBits(Handle)`. The memory block can be used to determine the size of the metafile or to save the metafile as a file.

See also: `::GetMetaFileBits`, `::SetMetaFileBits`

GetMetaFileBitsEx



```
inline DWORD GetMetaFileBitsEx(UINT size, void* data);
```

Retrieves the contents of the Windows-format metafile associated with this object and copies them (up to *size* bytes) to the *data* buffer. If *data* is nonzero and the call succeeds, the actual number of bytes copied is returned. If *data* is 0, a successful call returns the number of bytes required by the buffer. A return value of 0 always indicates a failure.

See also: `::GetMetaFileBitsEx`, `::SetMetaFileBitsEx`

IsOK

```
inline BOOL IsOK() const;
```

Returns TRUE if this object's Handle is nonzero, otherwise FALSE.

See also: `TMetaFilePict::Handle`

operator HMETAFILE()

```
inline operator HMETAFILE() const;
```

Type-conversion operator returning *Handle*.

See also: `TMetaFilePict::Handle`

TModule class

module.h

ObjectWindows dynamic-link libraries (DLLs) construct an instance of *TModule*, which acts as an object-oriented stand-in for the library (DLL) module. *TModule* defines behavior shared by both library and application modules. ObjectWindows applications construct an instance of *TApplication*, derived from *TModule*. *TModule*'s constructors manage loading and freeing of external DLLs, and the member functions provide support for default error handling.

Public data members

lpCmdLine

```
char far* lpCmdLine;
```

A null-terminated string, *lpCmdLine* points to a copy of the command-line arguments passed when the module is loaded. Notice that *lpCmdLine* is different from the WIN32 *lpCmdLine* in which the full path name of the module is appended to the command-line arguments. Whether running under WIN16 or WIN32, ObjectWindows *TModule::lpCmdLine* data member includes only the command-line arguments. Note that the run-time library global variables `_argv[]` and `_argc` contain identical information for both WIN16 and WIN32 APIs, and that `_argv[0]` points to the full path name of the module.

See also: *TApplication*

Module

```
extern TModule *Module;
```

Holds a global pointer to the current module.

Status

```
TStatus Status;
```

Status contains the module status and is included for backward compatibility with ObjectWindows 1.0 applications. ObjectWindows 2.0 instead uses exceptions to handle errors. Setting *Status* to any nonzero value will throw a *TXCompatibility* exception.

See also: *TXCompatibility::MapStatusCodeToString*

Public constructors and destructor

Constructor

```
TModule(const char far* name, HINSTANCE Instance, const char far*
        cmdLine);
```

Constructs a *TModule* object for an ObjectWindows DLL or program from within *LibMain* or *WinMain*. Calls *InitModule* to initialize *hInstance* and *cmdLine*.

Constructor

```
TModule(const char far* name, HINSTANCE hInstance);
```

Constructs a *TModule* object that is an alias for an already loaded DLL or program with an available *HInstance*. When the *TModule* is destructed, the instance isn't automatically freed. *name*, which is optional, can be 0.

Constructor

```
TModule(const char far* name, BOOL shouldLoad = TRUE);
```

Constructs a *TModule* object that is used as an alias for a DLL. If *shouldLoad* is TRUE, *Tmodule* will automatically load and free the DLL. If *shouldLoad* is FALSE, then the *HInstance* needs to be set later using *InitModule*.

Destructor

```
virtual ~TModule();
```

Destroys a *TModule* object and deletes *lpCmdLine*.

Public member functions

AccessResource



```
inline int AccessResource(HRSRC hRsrc) const;
```

Used for 16-bit applications, *AccessResource* finds the specified resource. The preferred method is to use *FindResource*.

See also: *TModule::FindResource*

AllocResource



```
inline HGLOBAL AllocResource(HRSRC hRsrc, DWORD size) const;
```

Used for 16-bit applications, *AllocResource* loads a resource into memory. The preferred method is to use *LoadResource*.

See also: *TModule::LoadResource*

CopyCursor



```
inline HCURSOR CopyCursor(HCURSOR hCursor) const;
```

Copies the cursor specified in *hCursor*. The return value is a handle to the duplicate cursor.

See also: *TIcon* class

CopyIcon

```
inline HICON CopyIcon(HICON hIcon) const;
```

CopyIcon copies the icon specified in *hIcon*. The return value is a handle to the icon or 0 if unsuccessful. When no longer required, the duplicate icon should be destroyed.

Error

```
virtual void Error(int errorCode);
```

Error processes errors identified by the error value supplied in *errorCode*. *Error* displays the error code in a message box and asks the user if it is OK to continue. If the user does continue, the program might or might not be able to recover. If the user does not continue, the program terminates. *Error* can be overridden with another kind of exception handler. This function is included only for backward compatibility with ObjectWindows 1.0. If you are writing ObjectWindows 2.0 applications, use the following *Error* function instead.

Error

```
virtual int Error(xmsg& x, unsigned captionResId, unsigned promptResId=0);
```

Called when fatal exceptions occur, *Error* takes an *xmsg* exception object, a resource ID for a message box caption, and an optional resource ID for a user prompt. By default, *Error* calls *HandleGlobalException* with the *xmsg* object and the strings obtained from the resources. An application (derived from *TApplication* which is derived from *TModule*) can reimplement this function to provide alternative behavior.

A non-zero status code is returned to indicate that an error condition is to be propagated; a zero status indicates that the condition has been handled

and that it is OK to proceed. ObjectWindows uses this status code inside its message loop to allow the program to resume. The global error handler (defined in `except.h`), which displays the message text, is

```
int _OWLFUNC HandleGlobalException(xmsg& x, char* caption, char*
                                canResume);
```

ExecDialog

```
int ExecDialog(TDialog* dialog);
```

Executes a dialog box. This function is included only for backward compatibility. Use *TDialog::Execute* instead.

FindResource

```
inline HRSRC FindResource(TResId id, const char far* type) const;
```

Finds the resource indicated by *id* and *type* and, if successful, returns a handle to the specified resource. If the resource cannot be found, the return value is zero. The *id* and *type* parameters either point to zero-terminated strings or specify an integer value. *type* can be one of the standard resource types (RT_ACCELERATOR, RT_BITMAP and so on). See the Windows API function *::FindResource* for a description of these values.

See also: *TInstance::AccessResource*, *TInstance::LoadResource*, *TInstance::SizeOfResource*

GetClientHandle

```
HWND GetClientHandle(HWND hWnd);
```

Gets the handle to the client window.

GetClassInfo

```
inline BOOL GetClassInfo(const char far* name, WNDCLASS far* wndclass)
                        const;
```

Used particularly for subclassing, *GetClassInfo* gets information about the window class specified in *wndclass*. *name* points to a 0-terminated string that contains the name of the class. *wndclass* points to the WNDCLASS structure that receives information about the class. See the Windows API online Help for a description of this structure. If successful, *GetClassInfo* returns nonzero. If a matching class cannot be found, *GetClassInfo* returns zero.

GetInstance

```
inline HINSTANCE GetInstance() const;
```

Returns the instance handle for this module.

GetInstanceData

```
inline int GetInstanceData(void* data, int len) const;
```

GetInstanceData gets data from an already running instance of an application. *len* is the size of the buffer.

**GetModuleFileName**

```
inline int GetModuleFileName(char far* buff, int maxChars);
```

Returns the expanded filename (path and filename) of the file from which the specified module was loaded. *buff* points to a buffer that holds the path

and file name. *maxChars* specifies the length of the buffer. The expanded filename is truncated if it exceeds this limit. *GetModuleFileName* returns 0 if an error occurs.

See also: *::GetModuleFileName*

GetModuleUsage



```
inline int GetModuleUsage() const;
```

Returns the reference count of the module, if successful. The reference count is incremented by one each time *GetModuleUsage* is called and decremented by one when *FreeLibrary* is called.

See also: *::GetModuleUsage*

GetName

```
inline const char far* GetName() const;
```

Gets the name of the module.

See also: *::GetName*

GetParentObject

```
TWindow* GetParentObject(HWND hWndParent);
```

Gets a handle to the parent window. This function is included only for backward compatibility with ObjectWindows 1.0.

See also: *::GetParentObject*

GetProcAddress

```
inline FARPROC GetProcAddress(const char far* fcnName) const;
```

Gets the address of an exported procedure or function.

See also: *::GetProcAddress*

InitModule

```
void InitModule(HINSTANCE Instance, const char far* cmdLine);
```

Performs any instance initialization necessary for the module. If the module can't be created, a *TXInvalidModule* exception is thrown.

IsLoaded

```
inline BOOL IsLoaded() const;
```

Returns nonzero if the instance handle is loaded. Use this function primarily to ensure that a given instance is loaded.

LoadAccelerators

```
inline HACCEL LoadAccelerators(TResId id) const;
```

Loads the accelerator table resource specified by *id*. *LoadAccelerators* loads the table only if it has not been previously loaded. If the table has already been loaded, *LoadAccelerators* returns a handle to the loaded table.

See also: *::LoadAccelerators*

LoadBitmap

```
inline HBITMAP LoadBitmap(TResId id) const;
```

Loads the bitmap resource specified by *id*. If the bitmap cannot be found, *LoadBitmap* returns 0.

LoadBitmap can be used to load predefined Windows bitmaps. In such cases, *id* must be one of the OBM_XXXX values (OBM_BTNCORNERS, OBM_BTFSIZE, and so on). See the Windows API online Help for a description of these values.

See also: *TBitMap* class, *::LoadBitmap*

LoadCursor

```
inline HCURSOR LoadCursor(TResId id) const
```

Loads the cursor resource specified by *id* into memory and returns a handle to the cursor resource. If the cursor resource cannot be found or identifies a resource that is not a cursor, *LoadCursor* returns 0.

LoadCursor can be used to load a predefined Windows cursor if the *id* is one of the IDC_XXXX values. See the Windows API online Help for a description of these values.

See also: *TCursor* class, *::LoadCursor*

LoadIcon

```
inline HICON LoadIcon(const char far* name) const;
```

Loads the icon resource indicated by the parameter, *name*, into memory. *LoadIcon* loads the icon only if it has not been previously loaded. If the icon resource cannot be found, *LoadIcon* returns 0.

LoadIcon can be used to load a predefined Windows cursor if *name* points to one of the IDI_XXXX values. See the Windows API online Help for a description of these values.

See also: *TIcon* class, *::LoadIcon*

LoadMenu

```
inline HMENU LoadMenu(TResId id) const;
```

Loads the menu resource indicated by *id* into memory. The menu resource is typically a collection of one or more MENUITEMTEMPLATE structures, which might contain one or more menu items. If the menu resource cannot be found, *LoadMenu* returns 0.

See also: *TMenu*, *::LoadMenu*

LoadResource

```
inline HGLOBAL LoadResource(HRSRC hRsrc) const;
```

Loads a resource indicated by *hRsrc* into memory and returns a handle to the memory block that contains the resource. If the resource cannot be found, the return value is 0. The *hRsrc* parameter must be a handle created by *FindResource*.

LoadResource loads the resource into memory only if it has not been previously loaded. If the resource has already been loaded, *LoadResource* increments the reference count by one and returns a handle to the existing resource. The resource remains loaded until it is discarded.

See also: *::LoadResource*

LoadString

```
inline int LoadString(UINT id, char far* buff, int maxChars) const;
```

Loads a string resource identified by *id* into the buffer pointed to by *buff*. *maxChars* indicates the size of the buffer to which the zero-terminated string is copied. A string longer than the length specified in *maxChars* is truncated. The return value is the number of characters copied into the buffer, or 0 if the string resource does not exist.

See also: *::LoadString*

LowMemory

```
inline BOOL LowMemory();
```

This function, which is obsolete, always returns 0.

MakeWindow

```
TWindow* MakeWindow(TWindow* win);
```

This function is obsolete. Use *TWindow's Create* function instead.

**operator
HINSTANCE**

```
inline operator HINSTANCE() const;
```

Returns the handle of the Windows application or DLL module represented by this *TModule*. The handle must be supplied as a parameter to Windows when loading resources.

RestoreMemory

```
inline void RestoreMemory();
```

This function, which is obsolete, restores memory.

SetInstance

```
void SetInstance(HINSTANCE hInstance);
```

Sets the instance handle for this *TModule*. *SetInstance* is used for special cases in which the *hInstance* is not known when the module is constructed.

SetResourceHandler

```
inline RSRCHDLRPROC SetResourceHandler(const char far* type, RSRCHDLRPROC  
loadProc) const;
```



Used for 16-bit applications, *SetResourceHandler* installs a callback function that loads resources. *type* points to a resource type. *loadProc* is the address of the callback procedure. If successful, *SetResourceHandler* returns a pointer to a previously installed resource handler. If no resource handler has been installed, *SetResourceHandler* returns a pointer to the default handler. This function is useful for handling user-defined resource types.

SizeOfResource

```
inline DWORD SizeofResource(HRSRC hRsrc) const;
```

Returns the size, in bytes, of the resource indicated by *hRsrc*. The resource must be a resource handle created by *FindResource*. If the resource cannot be found, the return value is 0.

Because of alignment in the executable file, the returned size might be larger than the actual size of the resource. An application cannot rely on *SizeofResource* for the exact size of a resource.

See also: *TInstance::AccessResource*, *TInstance::LoadResource*

ValidWindow

```
inline TWindow* ValidWindow(TWindow* win);
```

This function, which is obsolete, returns a handle to the valid window.

Protected data members

HInstance

```
HINSTANCE HInstance;
```

Contains the executing instance of either the Windows application or DLL module. The instance must be supplied as a parameter to Windows when loading resources.

Name

```
char far* Name;
```

Holds the name of the application or DLL module.

TModule::TXInvalidModule class**module.h**

A nested class, *TXInvalidModule* describes an exception that results from an invalid module. A window throws this exception if it can't create a valid *TModule* object.

Public constructors

Constructor

```
TXInvalidModule();
```

Constructs a *TXInvalidModule* object.

TOpenSaveDialog class**opensave.h**

TOpenSaveDialog is the base class for modal dialogs that let you open and save a file under a specified name. *TOpenSaveDialog* constructs a *TData* structure and passes it the *TOpenSaveDialog* constructor. Then the dialog is executed (modal) or created (modeless). Upon return, the necessary fields are updated including an error field that contains 0, or a common dialog extended error.

Public constructors

Constructor

```
TOpenSaveDialog(TWindow* parent, TData& data, TResID templateID = 0,
               const char far* title = 0, TModule* module = 0);
```

Constructs an open save dialog box object with the supplied parent window, data, resource ID, title, and current module object.

See also: *TOpenSaveDialog::TData struct*

Public member functions

GetFileName

```
inline static int GetFileName(const char far* fileName,
                             char far* fileName, int fileNameLen)
```

Stores the name of the file to be saved or opened.

GetFileNameLen

```
inline static int GetFileNameLen(const char far* fileName);
```

Stores the length of the name of the file to be saved or opened.

Protected data members

Data

```
TData& Data;
```

Stores the file name, its length, extension, filter, initial directory, default file-name extension, and any error messages.

ofn

```
OPENFILENAME ofn;
```

Contains the attributes of the file name such as length, extension, and directory. *ofn* is initialized using the fields in the *TOpenSaveDialog::TData* class.

See also: *TOpenSaveDialog::TData*

ShareViMsgId

```
static UINT ShareViMsgId;
```

Contains the message ID of the registered *ShareViolation* message.

See also: *TFileOpenSave::TData, TOpenSave::ShareViolation*

Protected constructors

Constructor

```
TOpenSaveDialog(TWindow* parent, TData& data, TModule* module);
```

Constructs a *TOpenSaveDialog* box object with the supplied parent, data, and current module object.

See also: *TOpenSaveDialog::TData struct*

Protected member functions

CmLbSelChanged

```
inline void CmLbSelChanged();
```

Indicates that the selection state of the file name list box in the *GetOpenFileName* or *GetSaveFileName* dialog boxes has changed. *CmLbSelChanged* is a default handler for command messages sent by *lst1* or *lst2* (the file and directory list boxes, respectively).

CmOk

```
inline void CmOk();
```

Responds to a click on the dialog box's OK button (with the identifier IDOK). Calls *CloseWindow* (passing IDOK).

See also: *TDialog::CloseWindow*

DialogFunction

```
BOOL DialogFunction(UINT message, WPARAM, LPARAM);
```

Returns TRUE if a message is handled, returns *ShareViMsgId* if a sharing violation occurs, otherwise returns FALSE.

See also: *TCommonDialog::DialogFunction*

DoExecute

```
int DoExecute() = 0;
```

Creates and executes a modal dialog box.

Init

```
void Init(TResID templateID);
```

Initializes a *TOpenSaveDialog* object with the current resource ID.

ShareViolation

```
virtual int ShareViolation();
```

If a sharing violation occurs when a file is opened or saved, *ShareViolation* is called to obtain a response. The default return value is OFN_SHAREWARN. Other Windows API sharing violation responses are listed in this table:

Constant	Meaning
OFN_SHAREFALLTHROUGH	Specifies that the file name can be used and that the dialog box should return it to the application.
OFN_OFN_SHARENOWARN	Instructs the dialog box to perform no further action with the file name and not to warn the user of the situation.
OFN_SHAREWARN	This is the default response that is defined as 0. Instructs the dialog box to display a standard warning message.

See also: *TFileOpenSave::TData*, *TOpenSave::ShareViMsgId*

Response table entries

The *TOpenSaveDialog* response table has no entries.

TOpenSaveDialog::TData struct

opensave.h

TOpenSaveDialog structure contains information about the user's file open or save selection.

Public data members

CustomFilter

char* CustomFilter;

CustomFilter stores the user-specified file filter; for example, *.CPP.

DefExt

char* DefExt;

DefExt stores the default extension.

Error

DWORD Error;

Error contains one or more of the following *CommDlgExtendedError* codes:

Constant	Meaning
CDERR_DIALOGFAILURE	Failed to create a dialog box.
CDERR_LOCKRESOURCEFAILURE	Failed to lock a specified resource.
CDERR_LOADRESFAILURE	Failed to load a specified resource.
CDERR_LOADSTRFAILURE	Failed to load a specified string.

Flags

DWORD Flags;

Flag contains one or more of the following Windows API constants:

Constant	Meaning
OFN_HIDEREADONLY	Hides the read-only check box.
OFN_FILEMUSTEXIST	Lets the user enter only names of existing files in the File Name entry field. If an invalid file name is entered, a warning message is displayed.
OFN_PATHMUSTEXIST	Lets the user enter only valid path names. If an invalid path name is entered, a warning message is displayed.
OFN_NOVALIDATE	Performs no check of the file name and requires the owner of a derived class to perform validation.
OFN_NOCHANGEDIR	Sets the current directory back to what it was when the dialog was initiated.
OFN_ALLOWMULTISELECT	Allows multiple selections in the File Name list box.
OFN_CREATEPROMPT	Asks if the user wants to create a file that does not currently exist.

Constant	Meaning
OFN_EXTENSIONDIFFERENT	Indicates the user entered a file name different from the specified in <i>DefExt</i> . This message is returned to the caller.
OFN_NOREADONLYRETURN	The returned file does not have the Read Only attribute set and is not in a write-protected directory. This message is returned to the caller.
OFN_NOTESTFILECREATE	The file is created after the dialog box is closed. If the application sets this flag, there is no check against write protection, a full disk, an open drive door, or network protection. For certain network environments, this flag should be set.
OFN_OVERWRITEPROMPT	The Save As dialog box displays a message asking the user if its OK to overwrite an existing file.
OFN_SHAREAWARE	If this flag is set and a call to open a file fails because of a sharing violation, the error is ignored and the dialog box returns the given file name. If this flag is not set, the virtual function <i>ShareViolation</i> is called, which returns OFN_SHAREWARN (by default) or one of the following values: OFN_SHAREFALLTHROUGH File name is returned from the dialog box. OFN_SHARENOWARN No further action is taken. OFN_SHAREWARN User receives the standard warning message for this type of error.
OFN_SHOWHELP	Shows the Help button in the dialog box.

FileName char* FileName;

Holds the name of the file to be saved or opened.

Filter char* Filter;

Filter holds the filter to use initially when displaying file names.

FilterIndex int FilterIndex;

FilterIndex indicates which filter to use initially when displaying file names.

InitialDir char* InitialDir;

InitialDir holds the directory to use initially when displaying file names.

Public constructors and destructor

Constructor TData(DWORD flags=0, char* filter=0, char* customFilter=0,
char* initialDir=0, char* defExt=0);

Constructs a TOpenSaveDialog::TData structure.

Destructor ~TData();

Destructs a TOpenSaveDialog::TData structure.

See also: *::CommDlgExtendedError, ::OpenFileName, TEditFile::FileData*

Public member functions

SetFilter

```
void SetFilter(const char*filter = 0);
```

Makes a copy of the filter list used to display the file names.

TOutputStream class

docview.h

Derived from *TStream* and *ostream*, *TOutputStream* is a base class used to create output storage streams for a document.

Public constructors

Constructor

```
TOutputStream(TDocument& doc, LPCSTR name, int mode);
```

Constructs a *TOutputStream* object. *doc* refers to the document object, *name* is the user-defined name of the stream, and *mode* is the mode of opening the stream.

See also: *TInStream*, *ofXXXXX document open enum*, *shdocument sharing enum*

TPaintDC class

dc.h

A DC class derived from *TWindowDC* that wraps begin and end paint calls for use in a WM_PAINT response function.

Public data members

Ps

```
PAINTSTRUCT Ps;
```

The paint structure associated with this *TPaintDC* object.

See also: *PAINSTRUCT*

Public constructors and destructor

Constructor

```
TPaintDC(HWND wnd);
```

Creates a *TPaintDC* object with the given owned window. The data member *Wnd* is set to *wnd*.

See also: *TWindowDC::Wnd*, *TDC::TDC*

Destructor

```
~TPaintDC();
```

Destroys this object.

TPalette class**gdiobjec.h**

TPalette is the GDI Palette class derived from *TGdiObect*. The *TPalette* constructors can create palettes from explicit information or indirectly from various color table types that are used by DIBs.

Public data members**enum TStockId**

```
enum TStockId{Default};
```

Enumerates the stock palette attributes.

Protected data members**Stocks[]**

```
static TPalette Stocks[];
```

*Note: This array no longer exists. Use *TDC::SelectStockObject* instead.*

The single static array of Windows stock palettes serving all *TPalette* objects. The stock palette is *DEFAULT_PALETTE*.

Public constructors**Constructor**

```
TPalette(HPALETTE handle, TAutoDelete autoDelete = NoAutoDelete);
```

Creates a *TPalette* object and sets the *Handle* data member to the given borrowed *handle*. The *ShouldDelete* data member defaults to *FALSE*, ensuring that the borrowed handle will not be deleted when the C++ object is destroyed.

See also: *TGdiObject::Handle*, *TGdiObject::ShouldDelete*

Constructor

```
TPalette(const TClipboard&);
```

Creates a *TPalette* object with values taken from the given Clipboard.

See also: *TClipboard::GetClipboardData*

Constructor

```
TPalette(const TPalette& palette);
```

This public copy constructor creates a complete copy of the given *palette* object, as in

```
TPalette myPalette = yourPalette;
```

See also: *TPalette::GetPaletteEntries*, *::CreatePalette*

Constructor

```
TPalette(const LOGPALETTE far* logPalette);
```

Creates a *TPalette* object from the given *logPalette* array. *Handle* is set via a WinAPI *CreatePalette(logPalette)* call.

See also: *::CreatePalette*

Constructor

```
TPalette(const PALETTEENTRY far* entries, int count);
```

Creates a *TPalette* object with *count* entries from the given *entries* array.

See also: *::CreatePalette*

Constructor

```
TPalette(const BITMAPINFO far* info, UINT flags = 0);
```

Win 3.0 DIB header only. Creates a *TPalette* object from the color table following the given *BITMAPINFO* structure. This constructor works only for 2-, 16-, and 256-color bitmaps. A 0 handle is returned for other bitmaps including 24-bit DIBs.

See also: *::CreatePalette*

Constructor

```
TPalette(const BITMAPCOREINFO far* core, UINT flags = 0);
```

Presentation Manager (PM) 1.x DIBs only. Creates a *TPalette* object from the color table following the given *BITMAPCOREINFO* structure. This constructor works only for 2-, 16-, and 256-color bitmaps. A 0 handle is returned for other bitmaps including 24-bit DIBs. Note that every color in a PM 1.x table must be present because there is no *ClrUsed* field in the DIB header.

See also: *::CreatePalette*

Constructor

```
TPalette(const TDib& dib, UINT flags = 0);
```

Creates a *TPalette* object from the given Win or PM DIB object. The *flags* argument represents the values of the Windows *Pe* data structure used to create the palette.

See also: *::CreatePalette*

Public member functions

AnimatePalette

```
inline void AnimatePalette(UINT start, UINT count,
                          const PALETTEENTRY far* entries);
```

Replaces entries in this logical palette from the *entries* array of PALETTEENTRY structures. *start* specifies the first entry to be animated, and *count* gives the number of entries to be animated. Windows maps the new entries into the system palette immediately.

See also: `::AnimatePalette`

```
GetNearestPaletteIndex inline UINT GetNearestPaletteIndex(TColor color) const;
```

Returns the index of the color entry that represents the best color in this palette to the given *color*.

See also: `::GetNearestPaletteIndex`, `TColor`

GetNumEntries

```
inline UINT GetNumEntries() const;
```

Returns the number of entries in this palette or 0 if the call fails.

See also: `TGdiObject::GetObject`

GetObject

```
inline BOOL GetObject(WORD far& numEntries) const;
```

Finds the number of entries in this logical palette and sets the value in the *numEntries* argument. To find the entire LOGPALETTE structure, use `GetPaletteEntries`. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: `TGdiObject::GetObject`, `TPalette::GetPaletteEntries`, `::GetObject`, struct LOGPALETTE

GetPaletteEntries

```
UINT GetPaletteEntries(WORD start, WORD count, PALETTEENTRY far* entries),
const;
```

Retrieves a range of entries in this logical palette, and places them in the *entries* array. *start* specifies the first entry to be retrieved, and *count* gives the number of entries to be retrieved. Returns the number of entries actually retrieved, or 0 if the call fails.

See also: `::GetPaletteEntries`, `TGdiObject::SetPaletteEntries`

GetPaletteEntry

```
inline UINT GetPaletteEntry(WORD index, PALETTEENTRY far& entry) const;
```

Retrieves the entry in this logical palette at *index*, and places it in the *entries* array. Returns the number of entries actually retrieved: 1 if successful or 0 if the call fails.

See also: `TPalette::SetPaletteEntry`, `::GetPaletteEntry`

TPalette class

GetStock

```
static TPalette& GetStock(TStockId id);
```

Provides access to stock Windows palette objects. Returns *TPalette::Stocks[id]*.

See also: *TPalette::Stocks[]*, enum *TStockId*.

operator <<

```
inline TClipboard& operator<< (TClipboard& clipboard, TPalette& palette);
```

Copies the given *palette* to the given *clipboard* argument. Returns a reference to the resulting Clipboard, which allows normal chaining of <<.

operator HPALETTE()

```
operator HPALETTE() const;
```

Typecasting operator. Converts this palette's *Handle* to type *HPALETTE*, which is the Windows data type representing the handle to a logical palette.

ResizePalette

```
inline BOOL ResizePalette(UINT numEntries);
```

Changes the size of this logical palette to the number given by *numEntries*. Returns TRUE if the call is successful; otherwise returns FALSE.

See also: *TPalette::AnimatePalette*

SetPaletteEntries

```
inline UINT SetPaletteEntries(WORD start, WORD count,  
                             const PALETTEENTRY far* entries);
```

Sets the RGB color values in this palette from the *entries* array of PALETTEENTRY structures. *start* specifies the first entry to be animated, and *count* gives the number of entries to be animated. Returns the number of entries actually set, or 0 if the call fails.

See also: *::SetPaletteEntries*, *TGdiObject::GetPaletteEntries*

SetPaletteEntry

```
inline UINT SetPaletteEntry(WORD index, const PALETTEENTRY far& entry);
```

Sets the RGB color value at *index* in this palette from the *entry* argument. *start* specifies the first entry to be animated, and *count* gives the number of entries to be animated. Returns 1 (the number of entries actually set if successful) or 0 if the call fails.

See also: *::SetPaletteEntry*, *TGdiObject::GetPaletteEntry*

ToClipboard

```
void ToClipboard(TClipboard& clipboard);
```

Moves this palette to the target *clipboard* argument. If a copy is to be put on the Clipboard, use *TPalette(myPalette).ToClipboard()*; to make a copy first. The handle in the temporary copy of the object will be moved to the Clipboard. *ToClipboard* should delete to FALSE so that the object on the Clipboard is not deleted. The handle will still be available for examination.

See also: *TClipboard::SetClipboardData*

UnrealizeObject

```
inline BOOL UnrealizeObject();
```

Directs the GDI to completely remap the logical palette to the system palette on the next *RealizePalette(HDC)* or *TDC::RealizePalette()* call. Returns TRUE if the call is successful; otherwise FALSE.

See also: *::UnrealizeObject*, *::RealizePalette*,

Protected member functions

Create

```
void Create(const BITMAPINFO far* info, UINT flags);
void Create(const BITMAPCOREINFO far* core, UINT flags);
```

Sets values in this palette from the given bitmap structure. These functions are usually called by the constructor rather than directly.

TPaletteEntry class

color.h

TPaletteEntry is a support class derived from the structure *tagPALETTEENTRY*. The latter is defined as follows:

```
typedef struct tagPALETTEENTRY {
    BYTE  peRed;
    BYTE  peGreen;
    BYTE  peBlue;
    BYTE  peFlags;
} PALETTEENTRY;
```

where *peRed*, *peGreen*, and *peBlue* specify the red, green, and blue intensity-values for a palette entry.

The *peFlags* member can have the following values:

Value	Meaning
PC_EXPLICIT	Specifies that the low-order word of the logical palette entry designates a hardware palette index. This flag allows the application to show the contents of the display device palette.
PC_NOCOLLAPSE	Specifies that the color be placed in an unused entry in the system palette instead of being matched to an existing color in the system palette. If there are no unused entries in the system palette, the color is matched normally. Once this color is in the system palette, colors in other logical palettes can be matched to this color.

Value	Meaning
PC_RESERVED	Specifies that the logical palette entry be used for palette animation; this prevents other windows from matching colors to this palette entry since the color frequently changes. If an unused system-palette entry is available, this color is placed in that entry. Otherwise, the color is available for animation.

TPaletteEntry is used in conjunction with the classes *TPalette* and *TColor* to simplify logical color-palette operations. Constructors are provided to create *TPaletteEntry* objects from explicit COLORREF and RGB values, or from *TColor* objects.

Public constructors

Constructor

```
TPaletteEntry(int r, int g, int b, int f = 0);
```

Creates a palette entry object with *peRed*, *peGreen*, *peBlue*, and *peFlags* set to *r*, *g*, *b*, and *f*, respectively.

See also: *tagPALETTEENTRY* struct

Constructor

```
TPaletteEntry(TColor c);
```

Creates a palette entry object with *peRed*, *peGreen*, *peBlue*, and *peFlags* set to *c.Red()*, *c.Green()*, *c.Blue()*, and *c.Flags*, respectively.

See also: *TColor::Red*, *TColor::Green*, *TColor::Blue*

TPen class

gdiobjec.h

TPen is derived from *TGdiObject*. It encapsulates the GDI pen tool. Pens can be constructed from explicit information or indirectly. *TPen* relies on the base destructor, *~TGdiObject*.

Public data members

enum TStockId

```
enum TStockId{Null, Black, White};
```

Enumerates the Windows stock pens.

Protected data members

Stocks[]

```
static class TPen Stocks[];
```

Note: This array no longer exists. Use `TDC::SelectStockObject` instead.

The single static array of Windows stock pen objects serving all *TPen* objects. The three stock pens are `NULL_PEN`, `BLACK_PEN`, and `WHITE_PEN`.

Public constructors

Constructor

```
TPen(HPEN handle, TAutoDelete autoDelete = NoAutoDelete);
```

Creates a *TPen* object and sets the *Handle* data member to the given borrowed *handle*. The *ShouldDelete* data member defaults to `FALSE`, ensuring that the borrowed handle will not be deleted when the C++ object is destroyed.

See also: *TGdiObject::Handle*, *TGdiObject::ShouldDelete*, *::CreatePenIndirect*

Constructor

```
TPen(TColor color, int width=1, int style=PS_SOLID);
```

Creates a *TPen* object with the given values. The *width* argument is in device units, but if set to 0, a 1-pixel width is assumed. Sets *Handle* via a Win API *CreatePen(style, width, color)* call with the given default values. If *color* is black or white, *width* is one, and *style* is solid, a stock pen handle is returned. The values for *style* are listed in the following table.

Value	Meaning
<code>PS_SOLID</code>	Creates a solid pen.
<code>PS_DASH</code>	Creates a dashed pen. Valid only when the pen width is one or less in device units.
<code>PS_DOT</code>	Creates a dotted pen. Valid only when the pen width is one or less in device units.
<code>PS_DASHDOT</code>	Creates a pen with alternating dashes and dots. Valid only when the pen width is one or less in device units.
<code>PS_DASHDOTDOT</code>	Creates a pen with alternating dashes and double-dots. Valid only when the pen width is one or less in device units.
<code>PS_NULL</code>	Creates a null pen.
<code>PS_INSIDEFRAME</code>	Creates a solid pen. When this pen is used in any GDI drawing function that takes a bounding rectangle, the dimensions of the figure will be shrunk so that it fits entirely in the bounding rectangle, taking into account the width of the pen.

See also: *::CreatePen*, *TColor*

Constructor

```
TPen(const LOGPEN far* logPen);
```

Creates a *TPen* object from the given *logPen* values. Sets *Handle* via a Win API *CreatePenIndirect(logPen)* call.

See also: *::CreatePenIndirect*

Constructor

```
TPen(DWORD penStyle, DWORD width, const TBrush& brush, DWORD styleCount,
     LPDWORD style);
```

Creates a *TPen* object with the given values. Sets *Handle* via a Win API *ExtCreatePen(penStyle, width, &logBrush, styleCount, style)* call (where *logBrush* reflects the value in the given *brush* object).

See also: *::ExtCreatePen*

Constructor

```
TPen(DWORD penStyle, DWORD width, const LOGBRUSH& logBrush,
     DWORD styleCount, LPDWORD style);
```

Creates a *TPen* object with the given values. Sets *Handle* via a Win API *ExtCreatePen(penStyle, width, &logBrush, styleCount, style)* call.

Public member functions

GetObject

```
inline BOOL GetObject(LOGPEN far& logPen) const;
```

Retrieves information about this pen object and places it in the given *LOGPEN* structure. Returns *TRUE* if the call is successful, otherwise *FALSE*.

See also: *TGdiObject::GetObject*, struct *LOGPEN*

GetStock

```
inline static TPen& GetStock(TStockId id);
```

Provides access to stock Windows pen objects. Returns *TPen::Stocks[id]*.

See also: *TPen::Stocks[]*, enum *TStockId*

operator HPEN()

```
operator HPEN() const;
```

Typecasting operator. Converts this pen's *Handle* to type *HPEN* (the Windows data type representing the handle to a logical pen).

TPicResult enum

validate.h

TPicResult is the result type returned by the *Picture* member function of *TPXPictureValidator*. It contains one of the following types: *prIncomplete*, *prEmpty*, *prError*, *prSyntax*, *prAmbiguous*, *prIncompNoFill*.

```
enum TPicResult;
```

See also: *TPXPictureValidator*

TPlacement enum

gadgetwi.h

Enumerates the placement of a gadget—either before or after another gadget.

```
enum TPlacement;
```

See also: *TGadgetWindow::PositionGadgets*

TPoint class

point.h

TPoint is a support class, derived from *tagPOINT*. Under WIN32, the latter is defined as

```
typedef struct tagPOINT {
    LONG x;
    LONG y;
} POINT;
```

Under WIN16, *tagPoint* is defined as

```
typedef struct tagPOINT {
    int x;
    int y;
} POINT;
```

TPoint encapsulates the notion of a two-dimensional point that usually represents a screen position. *TPoint* inherits two data members, the coordinates *x* and *y*, from *tagPOINT*. A *TPoint* object can be created from a pair of **ints**, from a point of type **POINT**, from a value of type **SIZE**, or from the low and high words of a **DWORD** value. Member functions and operators are provided for comparing, assigning, and manipulating points. Overloaded **<<** and **>>** operators are declared as friends of *TPoint*, allowing chained insertion and extraction of *TPoint* objects with streams.

Public constructors

Constructor

```
TPoint();
```

The default *TPoint* constructor.

TPoint class

Constructor

```
inline TPoint(int _x, int _y);
```

Creates a *TPoint* object with the given coordinates.

Constructor

```
inline TPoint(const POINT point);
```

Creates a *TPoint* object with $x = point.x$ and $y = point.y$.

See also: *POINT*

Constructor

```
inline TPoint(const SIZE size);
```

Creates a *TPoint* object with $x = size.cx$ and $y = size.cy$.

See also: *SIZE* struct

Constructor

```
inline TPoint(DWORD dw);
```

Creates a *TPoint* object with $x = LOWORD(dw)$ and $y = HIWORD(dw)$.

Public member functions

Offset

```
inline TPoint& Offset(int dx, int dy);
```

Offsets this point by the given delta arguments. This point is changed to $(x + dx, y + dy)$. Returns a reference to this point.

See also: *TPoint::OffsetBy*, *TPoint::operator+=*

OffsetBy

```
inline TPoint OffsetBy(int dx, int dy) const;
```

Calculates an offset to this point using the given displacement arguments. Returns the point $(x + dx, y + dy)$. This point is not changed.

See also: *TPoint::operator+*, *TPoint::Offset*

operator+

```
inline TPoint operator+(const TSize& size) const;
```

Calculates an offset to this point using the given *size* argument as the displacement. Returns the point $(x + size.cx, y + size.cy)$. This point is not changed.

See also: *TPoint::OffsetBy*, *TSize*

operator-

```
inline TPoint operator-(const TSize& size) const;
```

```
inline TSize operator-(const TPoint& point) const;
```

```
inline TPoint operator-() const;
```

The first version calculates a negative offset to this point using the given *size* argument as the displacement. Returns the point $(x - size.cx, y - size.cy)$. This point is not changed.

The second version calculates a distance from this point to the *point* argument. Returns the *TSize* object ($x - \text{point}.x, y - \text{point}.y$). This point is not changed.

The third version returns the point $(-x, -y)$. This point is not changed.

See also: *TPoint::operator+, TSize*

operator==

```
inline BOOL operator==(const TPoint& other) const;
```

Returns TRUE if this point is equal to the *other* point; otherwise returns FALSE.

See also: *TPoint::operator!=*

operator+=

```
inline TPoint& operator+=(const TSize& size) const;
```

Offsets this point by the given *size* argument. This point is changed to $(x + \text{size}.cx, y + \text{size}.cy)$. Returns a reference to this point.

See also: *TPoint::Offset, TPoint::operator-=, TSize*

operator-=

```
inline TPoint& operator-=(const TSize& size) const;
```

Negatively offsets this point by the given *size* argument. This point is changed to $(x - \text{size}.cx, y - \text{size}.cy)$. Returns a reference to this point.

See also: *TPoint::Offset, TPoint::operator+=, TSize*

operator!=

```
inline BOOL operator!=(const TPoint& other) const;
```

Returns FALSE if this point is equal to the *other* point; otherwise returns TRUE.

See also: *TPoint::operator==*

Friend functions

operator>>

```
friend inline ipstream& operator>>(ipstream& is, TPoint& p) const;
```

Extracts a *TPoint* object from *is*, the given input stream, and copies it to *p*. Returns a reference to the resulting stream, allowing the usual chaining of >> operations.

See also: *TPoint* **friend operator<<**, *class ipstream*

operator<<

```
friend inline opstream& operator<<(opstream& os, const TPoint& p) const;
```

Inserts the given *TPoint* object, *p*, into the *opstream*, *os*. Returns a reference to the resulting stream, allowing the usual chaining of << operations.

See also: *TPoint* **friend operator>>**, *opstream*

operator<<

```
friend inline ostream& operator<<(ostream& os, const TPoint& p) const;
```

Formats and inserts the given *TPoint* object, *p*, into the *ostream*, *os*. The format is "(x,y)". Returns a reference to the resulting stream, allowing the usual chaining of << operations.

See also: *TPoint* **friend operator>>**, *ostream*

TPointer<> class**point.h**

A small utility class, *TPointer*, provides automatic destruction for objects constructed using **new**. *TPointer* is a parameterized class that holds a pointer of its parameterized type and that overloads operators to behave like an object pointer. Once a pointer is assigned to a *TPointer* object, it is eventually be deleted, either when the function exits, an exception is thrown, or another pointer is assigned to the same object. A *TPointer* object can be instantiated using one of the following equivalent methods:

```
TPointer<SomeClass> p = new SomeClass;
TPointer<SomeClass> p(new SomeClass);
TPointer<SomeClass> p;    p = new SomeClass;
```

TPointer objects must be created on the stack; they can't be created with **new**. However, when `delete` is called on the *TPointer* object, the overloaded `delete` operator actually calls `delete` on the internal pointer. This process permits explicit destruction of the object that is pointed to. (Note that an ampersand must be used to provide `delete` with a pointer; for example, `delete &p`.)

Public constructors**Constructor**

```
inline TPointer() : TPointerBase<T>();
```

Default constructor in which *p* is initialized to 0.

Constructor

```
inline TPointer(T* pointer) : TPointerBase<T>(pointer);
```

Initialized constructor where *p* is initialized to *pointer*.

Public member functions**operator()**

```
inline operator T*();
```

The overloaded type conversion operator allows the *TPointer* object to be passed as a function argument or assigned to a variable as if it were a pointer.

operator =	<code>inline T* operator =(T* src);</code> Assignment operator <i>T*</i> is assigned to <i>p</i> .
operator =	<code>T* operator =(const TPointer<T>& src);</code> Assignment operator used when <i>r</i> is a const reference to <i>TPointer</i> of <i>T</i> . This operator saves converting if another pointer object is used.
operator !	<code>inline int operator !();</code> Zero test operator that tests for <i>p</i> .
operator ~	<code>inline void operator ~();</code> Overloaded complement operator.
operator delete	<code>inline void operator delete(void* p);</code> Overloaded delete operator.
T* operator	<code>inline T* operator->();</code> Provides access to the pointer.

TPopupMenu class

menu.h

TPopupMenu creates an empty pop-up menu to add to an existing window or pop-up menu. See Chapter 7 in the *ObjectWindows Programmer's Guide* for more information about menu objects.

Public constructors

Constructor	<code>TPopupMenu(TAutoDelete autoDelete = AutoDelete);</code> Constructs an empty pop-up menu.
--------------------	---

Public member functions

TrackPopupMenu	<code>inline BOOL TrackPopupMenu(UINT flags, int x, int y, int rsvd, HWND wnd, TRect* rect = 0);</code> Allows the application to create a pop-up menu at the specified location in the specified window. <i>flags</i> specifies a screen position and can be one of the
-----------------------	---

Windows API *TPM_xxxx* values (TPM_CENTERALIGN, TPM_LEFTALIGN, TPM_RIGHTALIGN, TPM_LEFTBUTTON, or TPM_RIGHTBUTTON). *wnd* is the handle to the window that receives messages about the menu. *x* specifies the horizontal position in screen coordinates of the left side of the menu. *y* specifies the vertical position in screen coordinates of the top of the menu (for example, 0,0 specifies that a menu's left corner is in the top left corner of the screen). *rect* defines the area that the user can click without dismissing the menu.

See also: *::TrackPopupMenu*

TrackPopupMenu

```
inline BOOL TrackPopupMenu(UINT flags, TPoint& point, int rsvd, HWND wnd,
                          TRect* rect = 0);
```

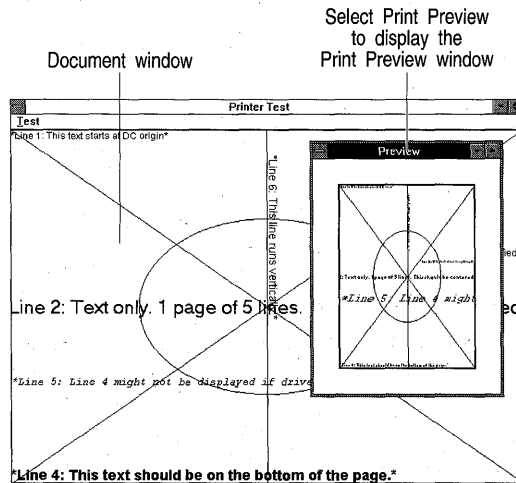
This function is the same as the previous *TrackPopupMenu* except that the *x* and *y* positions are specified in *point*.

See also: *::TrackPopupMenu*

TPreviewPage class

preview.h

TPreviewPage displays a page of a document in a print preview window. To obtain the information needed to display the page, *TPreviewPage* interacts with *TPrintPreviewDC* and *TPrintout*. Specifically, it creates a *TPrintPreviewDC* from the window DC provided in *Paint* and passes that *TPrintPreviewDC* object to *TPrintout*'s *SetPrintParams* member function. *PRINT.CPP*, in your OWLAPI\PRNTPREV directory, displays the following sample print preview window:



Public constructors

Constructor

```
PreviewPage(TWindow* parent, TPrintout& printout, TPrintDC& prndc,
            TSize& printExtent, int pagenum = 1);
```

Constructs a *TPreviewPage* object where *parent* is the parent window, *printout* is a reference to the corresponding *TPrintout* object, *prndc* is a reference to the *TPrintPreviewDC* object, *printExtent* is the extent (width and height) in logical units of the printed page, and *pagenum* is the number of the preview page. Initializes the style bits in *Attr.Style* so that *TPreviewPage* has the attributes of a visible child window with a thin border. Sets the background color of the preview page window to white.

Public member functions

Paint

```
void Paint(TDC& dc, BOOL, TRect& clip);
```

Displays the page in the preview window. To determine the preview page's attributes (line width, and so on), *Paint* calls several of *TPrintout*'s member functions. Then, to adjust the printer object for previewing, *Paint* determines if the page fits in the preview window or if clipping is necessary. Finally, *Paint* passes clipping and banding information to *TPrintout*'s *PrintPage* function, which is called to display the page in the preview window.

See also: *TPrintout::BeginPrinting*, *TPrintout::EndPrinting*, *TPrintout::PrintPage*

SetPageNumber

```
inline void SetPageNumber(int newNum);
```

Sets *newNum* to the number of the page currently displayed in the preview window.

Protected data members

PageNum

```
int PageNum;
```

Number of the page displayed in the preview window.

PrintDC

```
TPrintDC& PrintDC;
```

PrintDC& is a handle to the device context to use for printing.

PrintExtent

```
TSize PrintExtent;
```

Contains the extent (width and height) in logical units of the page.

Printout

TPrintout& Printout;

Holds a reference to the *TPrintout* object.**Protected member functions**

EvSize

void EvSize(UINT sizeType, TSize& size);

Invalidates the entire window when the size of the page displayed in the preview window changes.

See also: *TWindow::EvSize***Response table entries**

Response table entry	Member function
EV_WM_SIZE	EvSize

TPrintDC class**dc.h**Derived from *TDC*, *TPrintDC* provides access to a printer.**Public constructors**

Constructor

TPrintDC(HDC handle, TAutoDelete autoDelete = NoAutoDelete);

Creates a *TPrint* object for the DC given by *handle*.See also: *TAutoDelete*, *TDC::TDC***Constructor**

TPrintDC(const char far* driver, const char far* device, const char far* output, const DEVMODE far* initData);

Creates a *TPrint* object given print driver, device, output, and data from the DEVMODE structure.**Public member functions**

AbortDoc

inline int AbortDoc();

Aborts the current print job on this printer and erases everything drawn since the last call to *StartDoc*. *AbortDoc* calls the user-defined function set with *TPrintDC::SetAbortProc()* to abort a print job because of error or user intervention. *TPrintDC::EndDoc* should be used to terminate a successfully completed print job.

If successful, *AbortDoc* returns a positive or zero value; otherwise a negative value is returned.

Note that for pre-Win 3.1 applications, documents are aborted by calling *Escape()* with escape value ABORTDOC. For Win 3.1 and later, this escape method is superseded by the *::AbortDoc* function. *TPrintDC::AbortDoc* automatically selects the appropriate call.

See also: *::AbortDoc*, *TPrintDC::EndDoc*, *TPrintDC::SetAbortProc*, *TPrintDC::Escape*, ABORTDOC

BandInfo

```
inline int BandInfo(TBandInfo& bandInfo);
```

Retrieves information about the banding capabilities of this device, and copies it to the given *bandInfo* structure. Returns 1 if the call is successful; returns 0 if the call fails or if this device does not support banding.

See also: *struct TBandInfo*, *TPrintDC::Escape*, BANDINFO

DeviceCapabilities

```
static DWORD DeviceCapabilities(const char far* driver, const char far*
                               device, const char far* port,
                               int capability, char far* output=0,
                               LPDEVMODE devmode=0);
```

Retrieves data about the specified *capability* of the named printer *driver*, *device*, and *port*, and places the results in the *output* **char** array. The *driver*, *device*, and *port* names must be zero-terminated strings. The *devmode* argument points to a DEVMODE **struct**. If *devmode* is 0 (the default), *DeviceCapabilities* retrieves the current default initialization values for the specified printer driver; otherwise, it retrieves the values contained in the DEVMODE structure. The format of the *output* array depends on the capability being queried. If *output* is 0 (the default), *DeviceCapabilities* returns the number of bytes required in the *output* array. Possible values for *capability* are as follows:

Value	Meaning
DC_BINNAMES	The function enumerates the paper bins on the given device. If a device driver supports this constant, the <i>output</i> array is a data structure that contains two members. The first member is an array identifying valid paper bins:

```
short BinList[cBinMax]
```

The second member is an array of character strings specifying the bin names:

```
char PaperNames[cBinMax][cchBinName
```

If a device driver does not support this value, the *output* array is empty and the return value is NULL.

If *output* is NULL, the return value specifies the number of bins supported.

DC_BINS

The function retrieves a list of constants that identify the available bins and copies the list to the *output* array. If this array is NULL, the function returns the number of supported bins. The following bin identifiers can be returned:

```
DMBIN_AUTO
DMBIN_CASSETTE
DMBIN_ENVELOPE
DMBIN_ENVMANUAL
DMBIN_FIRST
DMBIN_LARGE CAPACITY
DMBIN_LARGE FMT
DMBIN_LAST
DMBIN_LOWER
DMBIN_MANUAL
DMBIN_MIDDLE
DMBIN_ONLY ONE
DMBIN_SMALL FMT
DMBIN_TRACTOR
DMBIN_UPPER
```

DC_DRIVER

The function returns the driver version number.

DC_DUPLEX

The function returns the level of duplex support. The return value is 1 if the function supports duplex output; otherwise it is 0.

DC_ENUMRESOLUTIONS

The function copies a list of available printer resolutions to the *output* array. The resolutions are copied as pairs of LONG integers; the first value of the pair specifies the horizontal resolution and the second value specifies the vertical resolution. If *output* is 0, the function returns the number of supported resolutions.

DC_EXTRA

The function returns the number of bytes required for the device-specific data that is appended to the DEVMODE structure.

DC_FIELDS

The function returns a value indicating which members of the DEVMODE structure are set by the device driver. This value can be one or more of the following constants:

```
DM_ORIENTATION
DM_PAPERSIZE
DM_PAPERLENGTH
DM_PAPERWIDTH
DM_SCALE
DM_COPIES
DM_DEFAULTSOURCE
DM_PRINTQUALITY
DM_COLOR
DM_DUPLEX
DM_YRESOLUTION
DM_TTOPTION
```

DC_FILEDEPENDENCIES

The function returns a list of files that must be loaded when the device driver is installed. If *output* is 0 and this value is specified, the function returns the number of file names that must be loaded. If *output* is nonzero, the function returns the specified number of 64-character file names.

DC_MAXEXTENT

The function returns the maximum supported paper-size. These dimensions are returned in a POINT structure; the *x* member gives the maximum paper width and the *y* member gives the maximum paper length.

DC_MINEXTENT

The function returns the minimum supported paper-size. These dimensions are returned in a POINT structure; the *x* member gives the minimum paper width and the *y* member gives the minimum paper length.

DC_PAPERS

The function retrieves a list of supported paper sizes and copies it to the *output* array. The function returns the number of sizes identified in the array. If *output* is 0, the function returns the number of supported paper sizes.

DC_PAPERSIZE

The function retrieves the supported paper sizes (specified in .1 millimeter units) and copies them to the *output* array.

DC_SIZE

The function returns the size of the DEVMODE structure required by the given device driver.

DC_VERSION

The function returns the device driver version number.

If *DeviceCapabilities* succeeds, the return value depends on the value of *capability*, as noted above. Otherwise, the return value is GDI_ERROR.

See also: *::DeviceCapabilitiesEx*, *struct DEVMODE*, *TDC::GetDeviceCaps*

EndDoc

```
inline int EndDoc();
```

Ends the current print job on this printer. *EndDoc* should be called immediately after a successfully completed print job. *TPrintDC::AbortDoc* should be used to terminate a print job because of error or user intervention.

If successful, *EndDoc* returns a positive or zero value; otherwise a negative value is returned.

For pre-Win 3.1 applications, documents are ended by calling *Escape()* with escape value ENDDOC. For Win 3.1 and later, this escape method is superseded by the *::EndDoc* function. *TPrintDC::EndDoc* automatically selects the appropriate call.

See also: *::EndDoc*, *TPrintDC::StartDoc*, *TPrintDC::AbortDC*, *TPrintDC::Escape*, *ABORTDOC*

EndPage

```
inline int EndPage();
```

Tells this printer's device driver that the application has finished writing to a page. If successful, *EndPage* returns a positive or zero value; otherwise a negative value is returned. Possible failure values are listed below:

Value	Meaning
SP_ERROR	General error.
SP_APPABORT	Job terminated because the application's print-canceling function returned 0.
SP_USERABORT	User terminated the job by using Windows Print Manager (PRINTMAN.EXE).

Value	Meaning
SP_OUTOFDISK	Insufficient disk space for spooling.
SP_OUTOFMEMORY	Insufficient memory for spooling.

For pre-Win 3.1 applications, page ends are signaled by calling *Escape()* with escape value *NEWFRAME*. For Win 3.1 and later, this escape method is superseded by the *::EndPage* function. *TPrintDC::EndPage* automatically selects the appropriate call.

See also: *::EndPage*, *TPrintDC::StartPage*, *TPrintDC::Escape*, *NEWFRAME*

Escape

```
inline int Escape(int escape, int count=0, const void* inData=0, void*
                outData=0);
```

Allows applications to access the capabilities of a particular device that are not directly available through the GDI of this DC. The *Escape* call is specified by setting a mnemonic value in the *escape* argument. In Win32 the use of *Escape* with certain *escape* values has been replaced by specific functions. The names of these new functions are based on the corresponding *escape* mnemonic, as shown in the following table:

Value	Action
ABORTDOC	Superseded by <i>TPrintDC::AbortDoc()</i> in Win32.
BANDINFO	Obsolete in Win32. Because all printer drivers for Windows version 3.1 and later set the text flag in every band, this escape is useful only for older printer drivers.
BEGIN_PATH	No changes for Win32. This escape is specific to PostScript printers.
CLIP_TO_PATH	No changes for Win32. This escape is specific to PostScript printers.
DEVICEDATA	Superseded in Win32. Applications should use the <i>PASSTHROUGH</i> escape to achieve the same functionality.
DRAFTMODE	Superseded in Win32. Applications can achieve the same functionality by setting the <i>dmPrintQuality</i> member of the <i>DEVMODE</i> structure to <i>DMRES_DRAFT</i> and passing this structure to the <i>CreateDC</i> function.
DRAWPATTERNRECT	No changes for Win32.
ENABLEDUPLEX	Superseded in Win32. Applications can achieve the same functionality by setting the <i>dmDuplex</i> member of the <i>DEVMODE</i> structure and passing this structure to the <i>CreateDC</i> function.

Value	Action
ENABLEPAIRKERNING	No changes for Win32.
ENBLERELATIVEWIDTHS	No changes for Win32.
ENDDOC	Superseded by <i>TPrintDC::EndDoc()</i> in Win32.
END_PATH	No changes for Win32. This escape is specific to PostScript printers.
ENUMPAPERBINS	Superseded in Win32. Applications can use <i>TPrintDC::DeviceCapabilities()</i> to achieve the same functionality.
ENUMPAPERMETRICS	Superseded in Win32. Applications can use <i>TPrintDC::DeviceCapabilities()</i> to achieve the same functionality.
EPSPRINTING	No changes for Win32. This escape is specific to PostScript printers.
EXT_DEVICE_CAPS	Superseded in Win32. Applications can use <i>TDC::GetDeviceCaps()</i> to achieve the same functionality. This escape is specific to PostScript printers.
EXTTEXTOUT	Superseded in Win32. Applications can use <i>TDC::ExtTextOut()</i> to achieve the same functionality. This escape is not supported by the version 3.1 PCL driver.
FLUSHOUTPUT	Removed for Win32.
GETCOLORTABLE	Removed for Win32.
GETEXTENDEDTEXTMETRICS	No changes for Win32. Support for this escape might change in future versions of Windows.
GETTEXTTABLE	Superseded in Win32. Applications can use <i>::GetCharWidth</i> to achieve the same functionality. This escape is not supported by the version 3.1 PCL or PostScript drivers.
GETFACENAME	No changes for Win32. This escape is specific to PostScript printers.
GETPAIRKERNTABLE	No changes for Win32.
GETPHYSPAGE SIZE	No changes for Win32. Support for this escape might change in future versions of Windows.
GETPRINTINGOFFSET	No changes for Win32. Support for this escape might change in future versions of Windows.
GETSCALINGFACTOR	No changes for Win32. Support for this escape might change in future versions of Windows.

Value	Action
GETSETPAPERBINS	Superseded in Win32. Applications can achieve the same functionality by calling <i>TPrintDC::DeviceCapabilities()</i> to find the number of paper bins, calling <i>::ExtDeviceMode</i> to find the current bin, and then setting the <i>dmDefaultSource</i> member of the DEVMODE structure and passing this structure to the <i>CreateDC</i> function. GETSETPAPERBINS changes the paper bin only for the current device context. A new device context will use the system-default paper bin until the bin is explicitly changed for that device context.
GETSETPAPERMETRICS	Obsolete in Win32. Applications can use <i>TPrintDC::DeviceCapabilities()</i> and <i>::ExtDeviceMode()</i> to achieve the same functionality.
GETSETPAPERORIENT	Obsolete in Win32. Applications can achieve the same functionality by setting the <i>dmOrientation</i> member of the DEVMODE structure and passing this structure to the <i>CreateDC</i> function. This escape is not supported by the Windows 3.1 PCL driver.
GETSETSCREENPARAMS	No changes for Win32.
GETTECHNOLOGY	No changes for Win32. Support for this escape might change in future versions of Windows. This escape is not supported by the Windows 3.1 PCL driver.
GETTRACKERNTABLE	No changes for Win32.
GETVECTORBRUSHSIZE	No changes for Win32. Support for this escape might change in future versions of Windows.
GETVECTORPENSIZE	No changes for Win32. Support for this escape might change in future versions of Windows.
MFCOMMENT	No changes for Win32.
NEWFRAME	No changes for Win32. Applications should use <i>::StartPage()</i> and <i>::EndPage()</i> instead of this escape. Support for this escape might change in future versions of Windows.
NEXTBAND	No changes for Win32. Support for this escape might change in future versions of Windows.
PASSTHROUGH	No changes for Win32.
QUERYESCAPESUPPORT	No changes for Win32.
RESTORE_CTM	No changes for Win32. This escape is specific to PostScript printers.
SAVE_CTM	No changes for Win32. This escape is specific to PostScript printers.

Value	Action
SELECTPAPERSOURCE	Obsolete in Win32. Applications can achieve the same functionality by using <i>TPrintDC::DeviceCapabilities()</i> .
SETABORTPROC	Superseded in Win32 by <i>::SetAbortProc()</i> . See <i>TPrintDC::SetAbortProc</i> .
SETALLJUSTVALUES	No changes for Win32. Support for this escape might change in future versions of Windows. This escape is not supported by the Windows 3.1 PCL driver.
SET_ARC_DIRECTION	No changes for Win32. This escape is specific to PostScript printers.
SET_BACKGROUND_COLOR	No changes for Win32. Applications should use <i>::SetBkColor()</i> instead of this escape. Support for this escape might change in future versions of Windows.
SET_BOUNDS	No changes for Win32. This escape is specific to PostScript printers.
SETCOLORTABLE	No changes for Win32. Support for this escape might change in future versions of Windows.
SETCOPYCOUNT	Superseded in Win32. An application should call <i>TPrintDC::DeviceCapabilities()</i> , specifying DC_COPIES for the nIndex parameter, to find the maximum number of copies the device can make. Then the application can set the number of copies by passing to the <i>CreateDC</i> function a pointer to the DEVMODE structure.
SETKERNTRACK	No changes for Win32.
SETLINECAP	No changes for Win32. This escape is specific to PostScript printers.
SETLINEJOIN	No changes for Win32. This escape is specific to PostScript printers.
SETMITERLIMIT	No changes for Win32. This escape is specific to PostScript printers.
SET_POLY_MODE	No changes for Win32. This escape is specific to PostScript printers.
SET_SCREEN_ANGLE	No changes for Win32.
SET_SPREAD	No changes for Win32.
STARTDOC	Superseded in Win32. Applications should call <i>::StartDoc</i> instead of this escape.
TRANSFORM_CTM	No changes for Win32. This escape is specific to PostScript printers.

Escape calls are translated and sent to the printer device driver. The *inData* buffer lets you supply any data needed for the escape. You must set *count* to the size (in bytes) of the *inData* buffer. If no input data is required, *inData* and *count* should be set to the default value of 0. Similarly, you must supply an *outData* buffer for those *Escape* calls that retrieve data. If the escape does not supply output, set *outData* to the default value of 0.

See also: *::Escape*

NextBand

```
int NextBand(TRect& rect);
```

Tells this printer's device driver that the application has finished writing to a band. The device driver sends the completed band to the Print Manager and copies the coordinates of the next band in the rectangle specified by *rect*.

If successful, *NextBand* returns a positive or zero value; otherwise a negative value is returned. Possible failure values are listed below:

Value	Meaning
SP_ERROR	General error.
SP_APPABORT	Job terminated because the application's print-canceling function returned 0.
SP_USERABORT	User terminated the job by using Windows Print Manager (PRINTMAN.EXE).
SP_OUTOFDISK	Insufficient disk space for spooling.
SP_OUTOFMEMORY	Insufficient memory for spooling.

See also: *TprintDC::Escape*, *NEXTBAND*, *TPrintDC::BandInfo*

QueryAbort

```
inline BOOL QueryAbort(int rsvd=0);
```



Tries to call the *AbortProc* callback function for this printer to determine if a print job should be aborted or not. *QueryAbort* returns the value returned by *AbortProc* or TRUE if no such callback function exists. TRUE indicates that printing should continue; FALSE indicates that the print job should be terminated. The *rsvd* argument is a reserved value that should be set to 0.

See also: *::QueryAbort*, *TPrintDC::SetAbortProc*, *TPrintDC::AbortDoc*, *::AbortProc*

QueryEscSupport

```
inline UINT QueryEscSupport(int escapeNum);
```

Returns TRUE if the escape specified by *escapeNum* is implemented on this device; otherwise FALSE.

See also: *TPrintDC::Escape*, *QUERYESCSUPPORT*

SetAbortProc

```
inline int SetAbortProc(ABORTPROC proc);
```

Establishes the user-defined *proc* as the printer-abort function for this printer. This function is called by *TPrintDC::AbortDoc* to cancel a print job during spooling.

Note that for pre-Win 3.1 applications, abort functions are set by calling *Escape()* with escape value SETABORTPROC. For Win 3.1 and later, this escape method is superseded by the *::SetAbortProc* function. *TPrintDC::SetAbortProc* automatically selects the appropriate call.

SetAbortProc returns a positive (nonzero) value if successful; otherwise it returns a negative (nonzero) value.

See also: *::SetAbortProc*, *TPrinterDC::Escape*, *SETABORTPROC*

SetCopyCount

```
inline int SetCopyCount(int requestCount, int& actualCount);
```

Sets to *requestCount* the number of uncollated copies of each page that this printer should print. The actual number of copies to be printed is copied to *actualCount*. The actual count will be less than the requested count if the latter exceeds the maximum allowed for this device. *SetCopyCount* returns 1 if successful; otherwise, it returns 0.

See also: *TPrintDC::DeviceCapabilities*, *TPrintDC::Escape*, *SETCOPYCOUNT*

StartDoc

```
inline int StartDoc(const char far* docName, const char far* output);
```

Starts a print job for the named document on this printer DC. If successful, *StartDoc* returns a positive value, the job ID for the document. If the call fails, the value SP_ERROR is returned. Detailed error information can be obtained by calling *GetLastError*.

This function replaces the earlier *::Escape* call with value STARTDOC.

See also: *::StartDoc*, *::GetJob*, *::SetJob*, *TPrintDC::EndDoc*, *TPrintDC::Escape*

StartPage

```
int StartPage();
```

Prepares this device to accept data. The system disables *::ResetDC()* between calls to *StartPage* and *EndPage*, so that applications cannot change the device mode except at page boundaries.

If successful, *StartPage* returns a positive value; otherwise, a negative or zero value is returned.

See also: *::StartPage*, *TPrintDC::EndPage*

Protected data members

DOCINFO

DOCINFO DocInfo;

Holds the input and output filenames used by *TPrintDC::StartDoc()*. The DOCINFO structure is defined as follows:

```
typedef struct {
    int cbSize;           // size of the structure, bytes
    LPSTR lpszDocName;   // document name <= 32 chars inc. final 0
    LPSTR lpszOutput;   // output file name
} DOCINFO;
```

The *lpszOutput* field allows a print job to be redirected to a file. If this field is NULL, the output will go to the device for the specified DC.

See also: *TPrintDC::StartDoc*

TPrintDialog class

printdia.h

TPrintDialog displays a modal print or a print setup dialog box. The print dialog box lets you specify the characteristics of a particular print job. The setup dialog box lets you configure the printer and specify additional print job characteristics.

Public constructors

Constructor

```
TPrintDialog(TWindow* parent, TData& data, const char far*
    printTemplateName=0, const char far* setupTemplateName=0,
    const char far* title=0, TModule* module=0);
```

Constructs a print or print setup dialog box with specified data, parent window, window caption, and module.

Public member functions

DoExecute

```
int DoExecute();
```

If no error occurs, *DoExecute* copies flags and print specifications into *data*. If an error occurs, *DoExecute* sets the error number of *data* to a *CommDlgExtendedError* code.

GetDefaultPrinter

```
BOOL GetDefaultPrinter();
```


Without displaying a dialog box, *GetDefaultPrinter* gets the device mode and name that are initialized for the system default printer.

Protected data members

Data

TData& Data;

Data is a reference to the *TData* object passed in the constructor.

See also: *TPrintDialog::TData*

pd

PRINTDLG pd;

Specifies the dialog box print job characteristics such as page range, number of copies, device context, and so on necessary to initialize the print or print setup dialog box. ObjectWindows passes this information to the Windows API *PrintDlg* function.

See also: *TPrintDialog::TData*

Protected member functions

CmSetup

inline void CmSetup();

Responds to the click of the setup button with an EV_COMMAND message.

DialogFunction

BOOL DialogFunction(UINT message, WPARAM, LPARAM);

Returns TRUE if a message is handled.

See also: *TDialog::DialogFunction*

Response table entries

The *TPrintDialog* response table has no entries.

TPrintDialog::TData struct

printdia.h

The *TPrintDialog* structure contains information necessary to initialize the dialog box with the user's printer selection.

Public data members

Copies

int Copies;

Copies indicates the number of copies to be printed.

Error

DWORD Error;

Error contains one or more of the following *CommDlgExtendedError* codes:

Constant	Meaning
CDERR_DIALOGFAILURE	Failed to create a dialog box.
CDERR_FINDRESFAILURE	Failed to find a specified resource.
CDERR_LOCKRESOURCEFAILURE	Failed to lock a specified resource.
CDERR_LOADRESFAILURE	Failed to load a specified resource.
CDERR_MEMALLOCFailure	Unable to allocate memory for internal data structures.
CDERR_MEMLOCKFAILURE	Unable to lock the memory associated with a handle.
PDERR_CREATEICFAILURE	<i>TPrintDialog</i> failed to create an information context.
PDERR_DEFAULTDIFFERENT	The printer described by structure members doesn't match the default printer. This error message can occur if the user changes the printer specified in the control panel.
PDERR_DNDMMISMATCH	The printer specified in <i>DevMode</i> and in <i>DevNames</i> is different.
PDERR_GETDEVMODEFAIL	The printer device-driver failed to initialize the <i>DevMode</i> structure.
PDERR_INITFAILURE	The <i>TPrintDialog</i> structure could not be initialized.
PDERR_LOADDRVFAILURE	The specified printer's device driver could not be loaded.
PDERR_NODEFAULTPRN	A default printer could not be identified.
PDERR_NODEVICES	No printer drivers exist.
PDERR_PARSEFAILURE	The string in the [devices] section of the WIN.INI file could not be parsed.
PDERR_PRINTERNOTFOUND	The [devices] section of the WIN.INI file doesn't contain the specified printer.
PDERR_RETDEFFAILURE	Either <i>DevMode</i> or <i>DevNames</i> contain zero.

Flags

DWORD Flags;

Flags can be one or more of the following Windows API constants:

Constant	Meaning
PD_ALLPAGES	Indicates that the All radio button was selected when the user closed the dialog box.
PD_COLLATE	Causes the Collate checkbox to be checked when the dialog box is created.
PD_DISABLEPRINTTOFILE	Disables the Print to File check box.
PD_HIDEPRINTTOFILE	Hides and disables the Print to File check box.
PD_NOPAGENUMS	Disables the Pages radio button and the associated edit control.
PD_NOSELECTION	Disables the Selection radio button.
PD_NOWARNING	Prevents the warning message from being displayed when there is no default printer.
PD_PAGENUMS	Selects the Pages radio button when the dialog box is created.
PD_PRINTSETUP	Displays the Print Setup dialog box rather than the Print dialog box.
PD_PRINTTOFILE	Checks the Print to File check box when the dialog box is created.
PD_RETURNDC	Returns a device context matching the selections that the user made in the dialog box.

Constant	Meaning
PD_RETURNDEFAULT	Returns <i>DevNames</i> structures that are initialized for the default printer without displaying a dialog box.
PD_RETURNIC	Returns an information context matching the selections that the user made in the dialog box.
PD_SELECTION	Selects the Selection radio button when the dialog box is created.
PD_SHOWHELP	Shows the Help button in the dialog box.
PD_USEDEVMODECOPIES	If a printer driver supports multiple copies, setting this flag causes the requested number of copies to be stored in the <i>dmCopies</i> member of the <i>DevMode</i> structure and 1 in <i>Copies</i> . If a printer driver does not support multiple copies, setting this flag disables the Copies edit control. If this flag is not set, the number 1 is stored in <i>DevMode</i> and the requested number of copies in <i>Copies</i> .

FromPage	int FromPage; <i>FromPage</i> indicates the beginning page to print.
MaxPage	int MaxPage; <i>MaxPage</i> indicates the maximum value for the beginning and ending pages to print.
MinPage	int MinPage; <i>MinPage</i> indicates the minimum value for the beginning and ending pages to print.
ToPage	int ToPage; <i>ToPage</i> indicates the ending page to print. See also: <code>::PRINTDLG</code>

Public member functions

ClearDevMode	void ClearDevMode(); Clears device mode information (information necessary to initialize the dialog controls).
ClearDevNames	void ClearDevNames(); Clears the device name information (information that contains three strings used to specify the driver name, the printer name, and the output port name).
GetDeviceName	const char far* GetDeviceName() const; Gets the name of the output device.

GetDevMode	inline const DEVMODE far* GetDevMode() const; Gets a pointer to a DEVMODE structure (a structure containing information necessary to initialize the dialog controls).
GetDevNames	inline const DEVNAMES far* GetDevNames() const; Gets a pointer to a DEVNAMES structure (a structure containing three strings used to specify the driver name, the printer name, and the output port name).
GetDriverName	const char far* GetDriverName() const; Gets the name of the printer device driver.
GetOutputName	const char far* GetOutputName() const; Gets the name of the physical output medium.
Lock	void Lock(); Locks memory associated with the DEVMODE and DEVNAMES structures.
SetDevMode	void SetDevMode(const DEVMODE far* devMode); Sets the values for the DEVMODE structure.
SetDevNames	void SetDevNames(const char far* driver, const char far* device, const char far* output); Sets the values for the DEVNAMES structure.
TransferDC	TPrintDC* TransferDC(); Creates and returns a <i>TPrintDC</i> with the current settings.
Unlock	void Unlock(); Unlocks memory associated with the DEVMODE and DEVNAMES structures.

TPrintPreviewDC class**preview.h**

Derived from *TPrintDC*, *TPrintPreviewDC* maps printer device coordinates to logical screen coordinates. It sets the extent of the view window and determines the screen and printer font attributes. Many of *TPrintPreviewDC*'s functions override *TDC*'s virtual functions.

Public constructors and destructor

Constructor

```
TPrintPreview(TDC& screen, TPrintDC& printdc, const TRect& client, const
             TRect& clip);
```

TPrintPreviewDC's constructor takes a screen DC as well as a printer DC. The screen DC is passed to the inherited constructor while the printer DC is copied to the member, *PrnDC*.

Destructor

```
~TPrintPreviewDC();
```

Destroys a *TPrintPreviewDC* object.

Public member functions

GetDeviceCaps

```
int GetDeviceCaps(int index) const;
```

GetDeviceCaps returns capability information, such as font and pitch attributes, about the printer DC. The *index* argument specifies the type of information required.

See also: *TDC::GetDeviceCaps*

LPToSDP

```
inline BOOL LPToSDP(TPoint* points, int count = 1) const;
```

```
inline BOOL LPToSDP(TRect& rect) const;
```

Converts each of the *count* points in the *points* array from logical points of the printer DC to screen points. Returns nonzero if the call is successful; otherwise, it returns 0.

See also: *TPrintPreviewDC::SDPtoLP*, *TDC::LPtoDP*

OffsetViewportOrg

```
BOOL OffsetViewportOrg(const TPoint& delta, TPoint far* oldOrg = 0);
```

Modifies this DC's viewport origin relative to the current values. The *delta* x- and y-components are added to the previous origin and the resulting point becomes the new viewport origin. The previous origin is saved in *oldOrg*. Returns nonzero if the call is successful; otherwise, returns 0.

See also: *TPrintPreviewDC::SetViewportOrg*, *TDC::OffsetViewportOrg*

ReOrg

```
virtual void ReOrg();
```

Gets the x- and y- extents of the viewport, equalizes the logical and screen points, and resets the x- and y- extents of the viewport.

ReScale

```
virtual void ReScale();
```

Maps the points of the printer DC to the screen DC. Sets the screen window extent equal to the maximum logical pointer of the printer DC.

RestoreFont void RestoreFont();

Restores the original GDI font object to this DC.

See also: *TPrintPreviewDC::SelectObject*, *TDC::OrgFont*

ScaleViewportExt BOOL ScaleViewportExt(int xNum, int xDenom, int yNum, int yDenom,
TSize far* oldExtent = 0);

Modifies this DC's viewport extents relative to the current values. The new extents are derived as follows:

$$xNewVE = (xOldVE * xNum) / xDenom$$

$$yNewVE = (yOldVE * yNum) / yDenom$$

The previous extents are saved in *oldExtent*. Returns nonzero if the call is successful; otherwise returns 0.

See also: *TDC::ScaleViewportExt*, *TPrintPreviewDC::SetViewportExt*

ScaleWindowExt BOOL ScaleWindowExt(int xNum, int xDenom, int yNum, int yDenom,
TSize far* oldExtent = 0);

Modifies this DC's window extents relative to the current values. The new extents are derived as follows:

$$xNewWE = (xOldWE * xNum) / xDenom$$

$$yNewWE = (yOldWE * yNum) / yDenom$$

The previous extents are saved in *oldExtent*. Returns nonzero if the call is successful; otherwise returns 0.

See also: *TDC::SetWindowExt*, *TPrintPreviewDC::ScaleWindowExt*

SDPtoLP inline BOOL SDPtoLP(TPoint* points, int count = 1) const;
inline BOOL SDPtoLP(TRect& rect) const;

Converts each of the *count* points in the *points* array from screen device points to logical points of the printer DC. *SDPtoLP* returns nonzero if the call is successful; otherwise, it returns 0.

See also: *TPrintPreviewDC::LPtoSDP*, *TDC::DPtoLP*

SelectObject void SelectObject(const TFont& newFont);

Selects the given font object into this DC.

See also: *TPrintPreviewDC::SelectStockObject*, *TDC::SelectObject*

SelectStockObject void SelectStockObject(int index);

Retrieves a handle to a predefined stock font.

See also: *TDC::SelectStockObject*

SetBkColor

`TColor SetBkColor(TColor color);`

Sets the current background color of this DC to the given *color* value or the nearest available. Returns 0x80000000 if the call fails.

See also: *TDC::SetBkColor*

SetMapMode

`int SetMapMode(int mode);`

Sets the current window mapping mode of this DC to *mode*. Returns the previous mapping mode value. The mapping mode defines how logical coordinates are mapped to device coordinates. It also controls the orientation of the device's x- and y-axes.

See also: *TDC::GetMapMode*, *TDC::SetMapMode*

SetTextColor

`TColor SetTextColor(TColor color);`

Sets the current text color of this DC to the given *color* value. The text color determines the color displayed by *TDC::TextOut* and *TDC::ExtTextOut*.

See also: *TDC::GetTextColor*, *TDC::SetTextColor*

SetViewportExt

`BOOL SetViewportExt(const TSize& extent, TSize far* oldExtent = 0);`

Sets the screen's viewport x- and y-extents to the given *extent* values. The previous extents are saved in *oldExtent*. Returns nonzero if the call is successful; otherwise, returns 0. The *extent* value determines the amount of stretching or compression needed in the logical coordinate system to fit the device coordinate system. *extent* also determines the relative orientation of the two coordinate systems.

See also: *TDC::GetViewportExt*, *TDC::SetViewportExt*

SetViewportOrg

`BOOL SetViewportOrg(const TPoint& origin, TPoint far* oldOrg=0);`

Sets the printer DC's viewport origin to the given *origin* value, and saves the previous origin in *oldOrg*. Returns nonzero if the call is successful; otherwise returns 0.

See also: *TPrintPreviewDC::OffsetViewportOrg*, *TDC::GetViewportOrg*, *TDC::SetViewportOrg*

SetWindowExt

`BOOL SetWindowExt(const TSize& extent, TSize far* oldExtent=0);`

Sets the DC's window x- and y-extents to the given *extent* values. The previous extents are saved in *oldExtent*. Returns nonzero if the call is successful; otherwise, returns 0. The *extent* value determines the amount of stretching or compression needed in the logical coordinate system to fit the

device coordinate system. *extent* also determines the relative orientation of the two coordinate systems.

See also: *TDC::GetWindowExt*, *TDC::SetWindowExt*, *TPrintPreviewDC::ScaleWindowExt*

SyncFont

virtual void SyncFont();

Sets the screen font equal to the current printer font.

Protected data members

CurrentPreviewFont TFont* CurrentPreviewFont;

The current view font.

PrnDC TPrintDC& PrnDC;

Holds a reference to the printer DC.

PrnFont HFONT PrnFont;

The current printer font.

Protected member functions

GetAttributeHDC HDC GetAttributeHDC() const;

Returns the attributes of the printer DC (*PrnDC*).

See also: *TDC::GetAttributeHDC*

TPrinter class**printer.h**

TPrinter represents the physical printer device. To print or configure a printer, initialize an instance of *TPrinter*.

Public constructors and destructor

Constructor TPrinter();

Constructs an instance of *TPrinter* associated with the default printer. To change the printer, call *SetDevice* after the object has been initialized or call *Setup* to let the user select the new device through a dialog box.

Destructor virtual ~TPrinter();

Frees the resources allocated to *TPrinter*.

Public member functions

ClearDevice

```
virtual void ClearDevice();
```

Called by *SetPrinter* and the destructor, *ClearDevice* disassociates the device with the current printer. *ClearDevice* changes the current status of the printer to `PF_UNASSOCIATED`, which causes the object to ignore all calls to *Print* until the object is reassociated with a printer.

Print

```
virtual BOOL Print(TWindow* parent, TPrintout& printout, BOOL prompt);
```

Print renders the given printout object on the associated printer device and displays an Abort dialog box while printing. It displays any errors encountered during printing.

See also: *TPrinter::Error*

ReportError

```
virtual void ReportError(TWindow* parent, TPrintout& printout);
```

Print calls *ReportError* if it encounters an error. By default, it brings up the system message box with an error string created from the default string table. This function can be overridden to show a custom error dialog box.

Setup

```
virtual void Setup(TWindow* parent);
```

Call this function when you want the user to select and/or configure the currently associated printer. *Setup* calls to return the printer setup dialog box that is shown to the user.

Protected data members

Data

```
TPrintDialog::TData* Data;
```

Data is a reference to the *TPrintDialog* data structure that contains information about the user's print selection.

See also: *TPrintDialog::TData* struct

Error

```
int Error;
```

Error is the error code returned by GDI during printing. This value is initialized during a call to *Print*.

Protected member functions

- CreateAbortWindow** virtual TWindow* CreateAbortWindow(TWindow* parent, TPrintout& printout);
Creates a printer abort dialog message box.
- ExecPrintDialog** virtual BOOL ExecPrintDialog(TWindow* parent);
Executes a *TPrintDialog*.
- GetDefaultPrinter** virtual void GetDefaultPrinter();
Updates the printer structure with information about the user's default printer.
- SetPrinter** inline void SetPrinter(const char* driver, const char* device, const char* output);
SetPrinter changes the printer device association. *Setup* calls *SetPrinter* to change the association interactively. The valid parameters to this method can be found in the [devices] section of the WIN.INI file.
Entries in the [devices] section have the following format:
`<device name>=<driver>, <port> {, <port>}`

TPrinterAbortDlg class

printer.h

TPrinterAbortDlg is the object type of the default printer-abort dialog box. This dialog box is initialized to display the title of the current printout, as well as the device and port currently used for printing.

TPrinterAbortDlg expects to have three static text controls, with control IDs of 101 for the title, 102 for the device, and 103 for the port. These controls must have "%s" somewhere in the text strings so that they can be replaced by the title, device, and port. The dialog-box controls can be in any position and tab order.

Public constructors

- Constructor** TPrinterAbortDlg(TWindow* parent, TResId resId, const char far* title, const char far* device, const char far* port);
Constructs an Abort dialog box that contains a Cancel button and displays the given title, device, and port.

Public member functions

EvCommand

virtual LRESULT EvCommand(UINT id, HWND hWndCtl, UINT notifyCode);

Handles the Cancel button on the Printer-abort dialog box.

SetupWindow

virtual void SetupWindow();

Associates objects with the dialog resource template so that the title, device, and port can be determined for printing. See the description of *TPrintoutFlags* for information about printing flags and printer status information.

See also: *TPrintoutFlags* enum

TPrintout class

printer.h

TPrintout represents the physical printed document that is to sent to a printer to be printed. *TPrintout* does the rendering of the document onto the printer. Because this object type is abstract, it cannot be used to print anything by itself. For every document, or document type, a class derived from *TPrintout* must be created and its *PrintPage* function must be overridden.

Public constructors and destructor

Constructor

TPrintout(const char far* title);

Constructs an instance of *TPrintOut* with the given title.

Destructor

virtual ~TPrintout();

Destroys the resources allocated by the constructor.

Public member functions

BeginDocument

virtual void BeginDocument(int startPage, int endPage, int flags);

The printer object's *Print* function calls *BeginDocument* once before printing each copy of a document. The *flags* field indicates if the current print band accepts graphics, text, or both.

The default *BeginDocument* does nothing. Derived objects can override *BeginDocument* to perform any initialization needed at the beginning of each copy of the document.

See also: *TPrintoutFlags* enum

BeginPrinting

```
virtual void BeginPrinting();
```

The printer object's *Print* function calls *BeginPrinting* once at the beginning of a print job, regardless of how many copies of the document are to be printed. Derived objects can override *BeginPrinting* to perform any initialization needed before printing.

EndDocument

```
virtual void EndDocument();
```

The printer object's *Print* function calls *EndDocument* after each copy of the document finishes printing. Derived objects can override *EndDocument* to perform any needed actions at the end of each document.

EndPrinting

```
virtual void EndPrinting();
```

The printer object's *Print* function calls *EndPrinting* after all copies of the document finish printing. Derived objects can override *EndPrinting* to perform any needed actions at the end of each document.

GetDialogInfo

```
virtual void GetDialogInfo(int& minPage, int& maxPage, int& selFromPage,
                           int& selToPage);
```

GetDialogInfo retrieves information needed to allow the printing of selected pages of the document and returns TRUE if page selection is possible. Use of page ranges is optional, but if the page count is easy to determine, *GetDialogInfo* sets the number of pages in the document. Otherwise, set the number of pages to 0; printing will continue until *HasPage* returns FALSE.

HasPage

```
virtual BOOL HasPage(int pageNumber);
```

HasPage is called after every page is printed. By default, it returns FALSE, indicating that only one page is to be printed. If the document contains more than one page, this function must be overridden to return TRUE while there are more pages to print.

PrintPage

```
virtual void PrintPage(int page, TRect& rect, unsigned flags);
```

PrintPage is called for every page (or band, if *Banding* is TRUE) and must be overridden to print the contents of the given page. The *rect* and *flags* parameters are used during banding to indicate the extent and type of band currently requested from the driver (and should be ignored if *Banding* is FALSE). *page* is the number of the current page.

SetPrintParams

```
virtual void SetPrintParams(TPrintDC* dc, TSize pageSize);
```

SetPrintParams sets *DC* to *dc* and *PageSize* to *pageSize*. The printer object's *Print* function calls *SetPrintParams* to obtain the information it needs to determine pagination and page count. Derived objects that override *SetPrintParams* must call the inherited function.

See also: *TPreviewPage::Paint*

Protected data members

Banding

BOOL Banding;

If *Banding* is TRUE, the printout is banded and the *PrintPage* function is called once for every band. Otherwise, *PrintPage* is called only once for every page. Banding a printout is more memory- and time-efficient than not banding. By default, *Banding* is set to FALSE.

DC

TPrintDC* DC;

DC is the handle to the device context to use for printing.

ForceAllBands

BOOL ForceAllBands;

Many device drivers do not provide all printer bands if both text and graphics are not performed on the first band (which is typically a text-only band). Leaving *ForceAllBands* TRUE forces the printer driver to provide all bands regardless of what calls are made in the *PrintPage* function. If *PrintPage* does nothing but display text, it is more efficient for *ForceAllBands* to be FALSE. By default, it is TRUE. *ForceAllBands* takes effect only if *Banding* is TRUE.

PageSize

TSize PageSize;

PageSize is the size of the print area on the printout page.

Title

char far* Title;

Title is the current title to use for the printout. By default, this title appears in the Abort dialog box and as the name of the job in the Print Manager.

TPrintoutFlags enum

printer.h

ObjectWindows defines the following banding constants used to set flags for printout objects.

Constant	Meaning
pfBoth	Current band accepts both text and graphics.
pfGraphics	Current band accepts only graphics.
pfText	Current band accepts only text.

See also: *TPrinter*, *TPrintOut*

TProcInstance class**point.h**

Designed for Win16 applications, *TProcInstance* handles creating and freeing an instance thunk. For Win32 applications, *TProcInstance* is non-functional. The address returned from *TProcInstance* can be passed as a parameter to call-back functions, window subclassing functions, or Windows dialog box functions. See the Windows API online Help for more information about *MakeProcInstance* and *FreeProcInstance*.

Public constructors and destructor**Constructor**

```
TProcInstance(FARPROC p);
```

Makes a *TProcInstance*, passing *p* as the address of the procedure. Under Win16, calls *::MakeProcInstance* to make an instance thunk for *p*. Under Win32, the constructor just saves *p*.

See also: *::MakeProcInstance*

Destructor

```
~TProcInstance()
```

Under WIN16, frees the instance thunk.

See also: *::FreeProcInstance*.

Public member functions**operator FARPROC**

```
operator FARPROC();
```

Under WIN16, returns the instance thunk. Under Win32, returns *p* from the constructor.

TPXPictureValidator class**validate.h**

TPXPictureValidator objects compare user input with a picture of a data format to determine the validity of entered data. The pictures are compatible with the pictures Borland's Paradox relational database uses to control data entry. For a complete description of picture specifiers, see the *Picture* member function.

Public constructors**Constructor**

```
TPXPictureValidator(const char far* pic, BOOL autoFill=FALSE);
```

Constructs a picture validator object by first calling the constructor inherited from *TValidator* and setting *pic* to point to it. Then sets the *voFill* bit in *Options* if *AutoFill* is TRUE and sets *Options* to *voOnAppend*. Throws a *TXValidator* exception if the picture is invalid.

Public member functions

Error

```
void Error();
```

Overrides *TValidator*'s virtual function and displays a message box that indicates an error in the picture format and displays the string pointed to by *Pic*.

See also: *TValidator::Error*

IsValid

```
BOOL IsValid(const char far* str);
```

IsValid overrides *TValidator*'s virtual function and compares the string passed in *str* with the format picture specified in *Pic*. *IsValid* returns TRUE if *Pic* is NULL or if *Picture* returns *Complete* for *str*, indicating that *str* needs no further input to meet the specified format; otherwise, it returns FALSE.

See also: *TPXPictureValidator::Picture*

IsValidInput

```
BOOL IsValidInput(char far* str, BOOL suppressFill);
```

IsValidInput overrides *TValidator*'s virtual function and checks the string passed in *str* against the format picture specified in *Pic*. *IsValid* returns TRUE if *Pic* is NULL or *Picture* does not return *Error* for *str*; otherwise, it returns FALSE. The *suppressFill* parameter overrides the value in *voFill* for the duration of the call to *IsValidInput*.

If *suppressFill* is FALSE and *voFill* is set, the call to *Picture* returns a filled string based on *str*, so the image in the edit control automatically reflects the format specified in *Pic*.

See also: *TPXPictureValidator::Picture*

Picture

```
virtual TPicResult Picture(char far* input, BOOL autoFill=FALSE);
```

Formats the string passed in *input* according to the format specified by the picture string pointed to by *Pic*. *Picture* returns *prError* if there is an error in the picture string or if *input* contains data that cannot fit the specified picture. Returns *prComplete* if *input* can fully satisfy the specified picture. Returns *prIncomplete* if *input* contains data that incompletely fits the specified picture.

The following characters are used in creating format pictures:

Table 1.28
Picture format
characters

Type of character	Character	Description	
Special	#	Accept only a digit	
	?	Accept only a letter (case_insensitive)	
	&	Accept only a letter, force to uppercase	
	@	Accept any character	
	!	Accept any character, force to uppercase	
Match	;	Take next character literally	
	*	Repetition count	
	[]	Option	
	{}	Grouping operators	
	,	Set of alternatives	
		All others	Taken literally

See also: *TPicResult* enum

Protected data member

Pic
string Pic;
Points to a string containing the picture that specifies the format for data in the associated edit control. The constructor sets *Pic* to a string that is passed as one of the parameters.

Protected member functions

CalcTerm
UINT CalcTerm(UINT termCh, UINT i);
Calculates the end of an input group without modifying it.

CheckComplete
TPicResult CheckComplete(UINT termCh, UINT& i, TPicResult rslt);
Checks *termCh* and returns *prAmbiguous* if the result is ambiguous.

Group
TPicResult Group(char far* input, UINT termCh, UINT& i, UINT& j);
Processes a picture group.

Iteration
TPicResult Iteration(char far* input, UINT termCh, UINT& i, UINT& j);
The input string (*input*) is repeated a specified number of times. *termCh* is the position of the last character in the string. *i* is the current index.

Process
TPicResult Process(char far* input, UINT termCh, UINT& i, UINT& j);
Calls *Scan* to scan the input string for a specified character.

Scan
TPicResult Scan(char far* input, UINT termCh, UINT& i, UINT& j);

Scans the input string for specified characters.

SkipToComma

```
BOOL SkipToComma(UINT termCh, UINT& i);
```

Finds the next comma separator in the edit control. Returns FALSE if the edit control doesn't contain a comma.

SyntaxCheck

```
BOOL SyntaxCheck();
```

Checks the picture string for the specified character. If the picture string is NULL, *SyntaxCheck* returns FALSE.

ToGroupEnd

```
void ToGroupEnd(UINT termCh, UINT& i);
```

Skips a character or a picture group specified in *termCh*. Characters specified include [,], {, }, ;, *.

TRadioButton class**radiobut.h**

A *TRadioButton* is an interface object that represents a corresponding radio button element in Windows. Use *TRadioButton* to create a radio button control in a parent *TWindow*. A *TRadioButton* can also be used to facilitate communication between your application and the radio button controls of a *TDialog*.

Radio buttons have two states: checked and unchecked. *TRadioButton* inherits its state management member functions from its base class, *TCheckBox*. Optionally, a radio button can be part of a group (*TGroupBox*) that visually and logically groups its controls. *TRadioButton* is a streamable class.

Public constructors**Constructor**

```
TRadioButton(TWindow* parent, int id, const char far* title, int x, int y,
             int w, int h, TGroupBox *group, TModule* module = 0);
```

Constructs a radio button object with the supplied parent window (*parent*), control ID (*id*), associated text (*title*), position (*x, y*) relative to the origin of the parent window's client area, width (*w*), height (*h*), and associated group box (*group*). Invokes the *TCheckBox* constructor with similar parameters. Then sets the *Attr.Style* data member to WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON.

See also: *TControl::TControl*

Constructor

```
TRadioButton(TWindow* parent, int resourceId, TGroupBox *group,
             TModule* module = 0);
```

Constructs a *TRadioButton* object to be associated with a radio button control of a *TDialog*. Invokes the *TCheckBox* constructor with identical parameters. The *resourceId* parameter must correspond to a radio button resource that you define.

Protected member functions

BNClicked

```
void BNClicked();
```

Responds to an incoming BN_CLICKED message.

Response table entries

Response table entry	Member function
EV_MESSAGE (BM_SETSTYLE, BMSetStyle)	BMSetStyle
EV_WM_GETDIGCODE	EvGetDigCode

TRangeValidator class

validate.h

A *TRangeValidator* object determines whether the data typed by a user falls within a designated range of integers. *TRangeValidator* is a streamable class.

Public constructors

Constructor

```
TRangeValidator(long min, long max);
```

Constructs a range validator object by first calling the constructor inherited from *TFilterValidator*, passing a set of characters containing the digits '0'..'9' and the characters '+' and '-'. Sets *Min* to *min* and *Max* to *max*, establishing the range of acceptable long integer values.

See also: *TFilterValidator::TFilterValidator*

Public member functions

Error

```
void Error();
```

Error overrides *TValidator*'s virtual function and displays a message box indicating that the entered value does not fall within the specified range.

IsValid

```
BOOL IsValid(const char far* str);
```

Converts the string *str* into an integer number and returns TRUE if the result meets all three of these conditions:

- It is a valid integer number
- Its value is greater than or equal to *min*
- Its value is less than or equal to *max*

If any of those tests fails, *IsValid* returns FALSE.

Transfer

```
UINT Transfer(char far* str, void* buffer, TTransferDirection direction);
```

Incorporates the three types, *tdSizeData*, *tdGetData*, and *tdSetData*, that a range validator can handle for its associated edit control. *str* is the edit control's string value, and *buffer* is the data passed to the edit control. Depending on the value of *direction*, *Transfer* either sets *str* from the number in *buffer* or sets the number at *buffer* to the value of the string *str*. If *direction* is *tdSetData*, *Transfer* sets *str* from *buffer*. If *direction* is *tdGetData*, *Transfer* sets *buffer* from *str*. If *direction* is *tdSizeData*, *Transfer* neither sets nor reads data.

Transfer always returns the size of the data transferred.

See also: *TWindow::Transfer*

Protected data members

Max

```
long Max;
```

Max is the highest valid long integer value for the edit control.

Min

```
long Min;
```

Min is the lowest valid long integer value for the edit control.

TRect class

point.h

TRect is a support class derived from *tagRect*. Under Win32, *tagRect* is defined as

```
typedef struct tagRECT {
    LONG left;
    LONG top;
    LONG right;
    LONG bottom;
} RECT;
```

Under Win16, *tagRect* is defined as

```
typedef struct tagRECT {
    int left;
    int top;
    int right;
    int bottom;
} RECT;
```

TRect encapsulates the properties of rectangles with sides parallel to the *x*- and *y*-axes. In *ObjectWindows*, these rectangles define the boundaries of windows, boxes, and clipping regions. *TRect* inherits four data members from *tagRect*: *left*, *top*, *right*, and *bottom*. These represent the top left and bottom right (*x*, *y*) coordinates of the rectangle. Note that *x* increases from left to right, and *y* increases from top to bottom.

TRect places no restrictions on the relative positions of top left and bottom right, so it is legal to have *left* > *right* and *top* > *bottom*. However, many manipulations—such as determining width and height, and forming unions and intersections—are simplified by *normalizing* the *TRect* objects involved. Normalizing a rectangle means interchanging the corner point coordinate values so that *left* < *right* and *top* < *bottom*. Normalization does not alter the physical properties of a rectangle. *myRect.Normalized()* creates normalized copy of *myRect* without changing *myRect*, while *myRect.Normalize()* changes *myRect* to a normalized format. Both members return the normalized rectangle.

TRect constructors are provided to create rectangles from either four **ints**, two *TPoint* objects, or one *TPoint* and one *TSize* object. In the latter case, the *TPoint* object specifies the top left point (also known as the rectangle's origin) and the *TSize* object supplies the width and height of the rectangle. Member functions perform a variety of rectangle tests and manipulations. Overloaded << and >> operators are declared as friends of *TRect*, allowing chained insertion and extraction of *TRect* objects with streams.

Public constructors

Constructor

```
TRect();
```

The default constructor.

Constructor

```
inline TRect(const RECT far& rect);
```

Copies the given *rect* to this object.

Constructor

```
inline TRect(int _left, int _top, int _right, int _bottom);
```

Creates a rectangle with the given values.

Constructor `inline TRect(const TPoint& upLeft, const TPoint& loRight);`
 Creates a rectangle with the given top left and bottom right points.

See also: *TPoint*

Constructor `inline TRect(const TPoint& origin, const TSize& extent);`
 Creates a rectangle with its origin (top left) at *origin*, width at *extent.cx*, and height at *extent.cy*.

See also: *TPoint*, *TSize*

Public member functions

Area `inline long Area() const;`
 Returns the area of this rectangle.

See also: *TRect::Size*

BottomLeft `inline TPoint BottomLeft() const;`
 Returns the *TPoint* object representing the bottom left corner of this rectangle.

See also: *TRect::TopLeft*, *TRect::TopRight*, *TRect::BottomRight*

BottomRight `inline const TPoint& BottomRight() const;`
`inline TPoint& BottomRight();`
 Returns the *TPoint* object representing the bottom right corner of this rectangle.

See also: *TRect::TopRight*, *TRect::BottomLeft*, *TRect::TopLeft*

Contains `inline BOOL Contains(const TPoint& point) const;`
`inline BOOL Contains(const TRect& other) const;`
 The first version returns TRUE if the given *point* lies within this rectangle; otherwise, it returns FALSE. If *point* is on the left vertical or on the top horizontal borders of the rectangle, *Contains* also returns TRUE, but if *point* is on the right vertical or bottom horizontal borders, *Contains* returns FALSE.

The second version returns TRUE if the other rectangle lies on or within this rectangle; otherwise, it returns FALSE.

See also: *TRect::Touches*

Height `inline int Height() const;`

Returns the height of this rectangle (*bottom - top*).

See also: *TRect::Width*

InflatedBy

```
inline TRect InflatedBy(int dx, int dy) const;
inline TRect InflatedBy(const TSize& size) const;
```

Returns a rectangle inflated by the given delta arguments. In the first version, the top left corner of the returned rectangle is (*left - dx, top - dy*), while its bottom right corner is (*right + dx, bottom + dy*). In the second version the new corners are (*left - size.cx, top - size.cy*) and (*right + size.cx, bottom + size.cy*). The calling rectangle object is unchanged.

See also: *TRect::OffsetBy, TSize*

IsEmpty

```
inline BOOL IsEmpty() const;
```

Returns TRUE if *left >= right* or *top >= bottom*; otherwise, returns FALSE.

See also: *TRect::SetEmpty, TRect::IsNull*

IsNull

```
inline BOOL IsNull() const;
```

Returns TRUE if *left, right, top, and bottom* are all 0; otherwise, returns FALSE.

See also: *TRect::IsEmpty, TRect::SetEmpty*

Normalize

```
inline TRect& Normalize();
```

Normalizes this rectangle by switching the *left* and *right* data member values if *left > right*, and switching the *top* and *bottom* data member values if *top > bottom*. *Normalize* returns the normalized rectangle. A valid but nonnormal rectangle might have *left > right* and/or *top > bottom*. In such cases, many manipulations (such as determining width and height) become unnecessarily complicated. Normalizing a rectangle means interchanging the corner point values so that *left < right* and *top < bottom*. The physical properties of a rectangle are unchanged by this process.

See also: *TRect::Normalized*

Normalized

```
inline TRect Normalized() const;
```

Returns a normalized rectangle with the top left corner at (*Min(left, right), Min(top, bottom)*) and the bottom right corner at (*Max(left, right), Max(top, bottom)*). The calling rectangle object is unchanged. A valid but nonnormal rectangle might have *left > right* and/or *top > bottom*. In such cases, many manipulations (such as determining width and height) become unnecessarily complicated. Normalizing a rectangle means interchanging the corner point values so that *left < right* and *top < bottom*. The physical properties of a rectangle are unchanged by this process.

Note that many calculations assume a normalized rectangle. Some Windows API functions behave erratically if an inside-out *Rect* is passed.

See also: *TRect::Normalize*

Offset

```
inline TRect& Offset(int dx, int dy);
```

Changes this rectangle so its corners are offset by the given delta values. The revised rectangle has a top left corner at $(left + dx, top + dy)$ and a bottom right corner at $(right + dx, bottom + dy)$. The revised rectangle is returned.

See also: *TRect::operator+*, *TRect::operator+=*, *TRect::OffsetBy*

OffsetBy

```
inline TRect OffsetBy(int dx, int dy) const;
```

Returns a rectangle with the corners offset by the given delta values. The returned rectangle has a top left corner at $(left + dx, top + dy)$ and a bottom right corner at $(right + dx, bottom + dy)$.

See also: *TRect::operator+*

operator+

```
inline TRect operator+(const TSize& size) const;
```

Returns a rectangle offset positively by the delta values given *size*. The returned rectangle has a top left corner at $(left + size.x, top + size.y)$ and a bottom right corner at $(right + size.x, bottom + size.y)$. The calling rectangle object is unchanged.

See also: *TRect::OffsetBy*, *TSize*

operator-

```
inline TRect operator-(const TSize& size) const;
```

Returns a rectangle offset negatively by the delta values given *size*. The returned rectangle has a top left corner at $(left - size.cx, top - size.cy)$ and a bottom right corner at $(right - size.cx, bottom - size.cy)$. The calling rectangle object is unchanged.

See also: *TRect::OffsetBy*, *TSize*

operator&

```
inline TRect operator&(const TRect& other) const;
```

Returns the intersection of this rectangle and the *other* rectangle. The calling rectangle object is unchanged. Returns a NULL rectangle if the two don't intersect.

See also: *TRect::operator|*, *TRect::operator&=*

operator|

```
inline TRect operator|(const TRect& other) const;
```

Returns the union of this rectangle and the *other* rectangle. The calling rectangle object is unchanged.

See also: `TRect::operator&`, `TRect::operator|`

operator== inline BOOL operator==(const TRect& other) const;

Returns TRUE if this rectangle has identical corner coordinates to the *other* rectangle; otherwise, returns FALSE.

See also: `TRect::operator!=`

operator!= inline BOOL operator!=(const TRect& other) const;

Returns FALSE if this rectangle has identical corner coordinates to the *other* rectangle; otherwise, returns TRUE.

See also: `TRect::operator==`

operator+= inline TRect& operator+=(const TSize& delta);

Changes this rectangle so its corners are offset by the given delta values, *delta.x* and *delta.y*. The revised rectangle has a top left corner at (*left + delta.x*, *top + delta.y*) and a bottom right corner at (*right + delta.x*, *bottom + delta.y*). The revised rectangle is returned.

See also: `TRect::operator+`, `TRect::OffsetBy`, `TRect::Offset`

operator-= inline TRect& operator-=(const TSize& delta);

Changes this rectangle so its corners are offset negatively by the given delta values, *delta.x* and *delta.y*. The revised rectangle has a top left corner at (*left - delta.x*, *top - delta.y*) and a bottom right corner at (*right - delta.x*, *bottom - delta.y*). The revised rectangle is returned.

See also: `TRect::operator-`, `TRect::operator+=`, `TRect::OffsetBy`, `TRect::Offset`

operator&= inline TRect& operator&=(const TRect& other) const;

Changes this rectangle to its intersection with the *other* rectangle. This rectangle object is returned. Returns a NULL rectangle if there is no intersection.

See also: `TRect::operator&`, `TRect::operator|`

operator|= inline TRect& operator|=(const TRect& other) const;

Changes this rectangle to its union with the *other* rectangle. This rectangle object is returned.

See also: `TRect::operator|`, `TRect::operator&=`

operator TPoint*() inline operator const TPoint*() const;
inline operator TPoint*()

Type conversion operators converting the pointer to this rectangle to type pointer to *TPoint*.

See also: class *TPoint*

Set

```
inline void Set(int _left, int _top, int _right, int _bottom);
```

Repositions and resizes this rectangle to the given values.

SetEmpty

```
inline void SetEmpty();
```

Empties this rectangle by setting *left*, *top*, *right*, and *bottom* to 0.

Size

```
inline TSize Size() const;
```

Returns a *TSize* object representing the width and height of this rectangle.

See also: *TSize*

TopLeft

```
inline const TPoint& TopLeft() const;
```

```
inline TPoint& TopLeft();
```

Returns the *TPoint* object representing the top left corner of this rectangle.

See also: *TRect::TopRight*, *TRect::BottomLeft*, *TRect::BottomRight*

TopRight

```
inline TPoint TopRight() const;
```

Returns the *TPoint* object representing the top right corner of this rectangle.

See also: *TRect::TopLeft*, *TRect::BottomLeft*, *TRect::BottomRight*

Touches

```
inline BOOL Touches(const TRect& other) const;
```

Returns TRUE if the *other* rectangle shares any interior points with this rectangle; otherwise, returns FALSE.

See also: *TRect::Contains*

Width

```
inline int Width() const;
```

Returns the width of this rectangle (*right* – *left*).

See also: *TRect::Height*

Friend functions

operator>>

```
friend inline ipstream& operator>>(ipstream& is, TRect& r) const;
```

Extracts a *TRect* object from *is*, the given input stream, and copies it to *r*. Returns a reference to the resulting stream, allowing the usual chaining of >> operations.

See also: *TRect* **friend operator<<**, *ipstream*

operator<<

```
friend inline opstream& operator<<(opstream& os, const TRect& r) const;
```

Inserts the given *TRect* object, *r*, into the *ostream*, *os*. Returns a reference to the resulting stream, allowing the usual chaining of << operations.

See also: *TRect* **friend operator>>**, *ostream*

operator<<

```
friend inline ostream& operator<<(ostream& os, const TRect& r) const;
```

Formats and inserts the given *TRect* object, *r*, into the *ostream*, *os*. The format is *(r.left, r.top)(r.right, r.bottom)*. Returns a reference to the resulting stream and allows the usual chaining of << operations.

See also: *TRect* **friend operator>>**, *ostream*

TRegion class**gdiobjec.h**

TRegion, derived from *TGdiobject*, represents GDI abstract shapes or regions. *TRegion* can construct region objects with various shapes. Several operators are provided for combining and comparing regions.

Public data members**enum TEllipse**

```
enum TEllipse{Ellipse};
```

Defines the class-specific constant, *Ellipse*, used to distinguish the ellipse constructor from the rectangle copy constructor.

See also: *TRegion::TRegion(const TRect& rect)*, *TRegion::TRegion(const TRect& E, TEllipse)*

Public constructors**Constructor**

```
TRegion();
```

The default constructor creates an empty *TRegion* object. *Handle* is set to 0 and *ShouldDelete* is set to TRUE.

See also: *TGdiObject::Handle*, *TGdiObject::ShouldDelete*, *::CreateRectRgn*

Constructor

```
TRegion(HRGN handle, TAutoDelete autoDelete = NoAutoDelete);
```

Creates a *TRegion* object and sets the *Handle* data member to the given borrowed *handle*. The *ShouldDelete* data member defaults to FALSE, ensuring that the borrowed handle is not deleted when the C++ object is destroyed. *HRGN* is the Windows data type representing the handle to an abstract shape.

Constructor TRegion(const TRegion& region);

This public copy constructor creates a copy of the given *TRegion* object, as in:

```
TRegion myRegion = yourRegion;
```

See also: *::CreateRectRgn*, *::CombineRgn*

Constructor TRegion(const TRect& rect);

Creates a region object from the given *TRectangle* object, as in:

```
TRegion myRegion(rect1);  
TRegion* pRegion;  
pRegion = new TRegion(rect2);
```

See also: *::CreateRectRgnIndirect*, *TRect*

Constructor TRegion(const TRect& E, TEllipse);

Creates the elliptical *TRegion* object that inscribes the given rectangle *E*. The *TEllipse* argument distinguishes this constructor from the *TRegion(const TRect& rect)* constructor.

See also: *::CreateEllipticRgnIndirect*, *TRect*

Constructor TRegion(const TRect& rect, const TSize& corner);

Creates a *TRegion* object from the given *rect* and *corner*.

Constructor TRegion(const TPoint* points, int count, int fillMode);

Creates a filled *TRegion* object from the polygons given by *points* and *fillMode*.

See also: *::CreatePolygonRgn*, *TPoint*

Constructor TRegion(const TPoint* points, const int* polyCounts, int count, int fillMode);

Creates a filled *TRegion* object from the polygons given by *points* and *fillMode*.

See also: *::CreatePolyPolygonRgn*, *TPoint*

Public member functions

Contains inline BOOL Contains(const TPoint& point) const;

Returns TRUE if this region contains the given point.

See also: `::PtInRegion`, `TPoint`

GetRgnBox

```
inline int GetRgnBox(TRect& box) const;
inline TRect GetRgnBox() const;
```

Finds the bounding rectangle (the minimum rectangle containing this region). In the first version, the resulting rectangle is placed in *box* and the returned values are as follows:

Value	Meaning
COMPLEXREGION	Region has overlapping borders.
NULLREGION	Region is empty.
SIMPLEREGION	Region has no overlapping borders.

In the second version, the resulting rectangle is returned.

See also: `::GetRgnBox`, `TRect`

operator==

```
inline BOOL operator==(const TRegion& other) const;
```

Returns TRUE if this region is equal to the *other* region.

See also: `::EqualRgn`, `TRegion::operator!=`

operator !=

```
inline BOOL operator!=(const TRegion& other) const;
```

Returns TRUE if this region is not equal to the *other* region.

See also: `::EqualRgn`, `TRegion::operator==`

operator=

```
TRegion& operator=(const TRegion& source);
```

Assigns the *source* region to this region. A reference to the result is returned, allowing chained assignments.

See also: `::CombineRgn`

operator +=

```
TRegion& operator+=(const TSize& delta);
```

Adds the given *delta* to each point of this region to displace (translate) it by *delta.x* and *delta.y*. Returns a reference to the resulting region.

See also: `::OffsetRgn`, `TSize`, `TRegion::operator-=`

operator -=

```
TRegion& operator-=(const TSize& delta);
TRegion& operator-=(const TRegion& source);
```

The first version subtracts the given *delta* from each point of this region to displace (translate) it by $-\text{delta.x}$ and $-\text{delta.y}$. The second version uses the RGN_DIF argument in `::CombineRegion` to create a “difference” region consisting of all parts of this region that are not parts of the *source* region. Both versions returns a reference to the resulting region.

See also: `::OffsetRgn`, `::CombineRgn`, `TSize`, `TRegion::operator+=`, `RGN_DIF`.

operator &=

```
TRegion& operator&=(const TRegion& source);
TRegion& operator&=(const TRect& source);
```

Creates the intersection of this region with the given *source* region or rectangle, and returns a reference to the result.

See also: `::CombineRgn`, `TRect`

operator |=

```
TRegion& operator|=(const TRegion& source);
TRegion& operator|=(const TRect& source);
```

Creates the union of this region and the given *source* region or rectangle, and returns a reference to the result.

See also: `::CombineRgn`, `TRect`

operator ^=

```
TRegion& operator^=(const TRegion& source);
TRegion& operator^=(const TRect& source);
```

Creates the union of this region and the given *source* region or rectangle, but excludes any overlapping areas. Returns a reference to the resulting region object.

See also: `::CombineRgn`, `TRect`

operator HRGN()

```
inline operator HRGN() const;
```

Typecast operator. *HRGN* is the Windows data type representing the handle to a physical region.

SetRectRgn

```
inline void SetRectRgn(const TRect& rect);
```

Uses the Win API *SetRectRgn* function to creates a rectangle of the size given by *rect*. (Does not use the local memory manager.)

See also: `::SetRectRgn`, `TRect`

Touches

```
inline BOOL Touches(const TRect& rect) const;
```

Returns TRUE if this region touches the given rectangle.

See also: `::RectInRegion`, `TRect`

TRelationship enum

layoutco.h

```
enum TRelationship;
```

Used by the *TLayoutConstraint* struct, *TRelationship* specifies the relationship between the edges and sizes of one window and the edges and sizes of another window (which can be a parent or sibling). These relationships can be specified as either the same value as the sibling or parent window (*ImAsIs*), an absolute value (*ImAbsolute*), a percent of one of the windows (*ImPercentOf*), a value that is either added above (*ImAbove*) or left (*ImLeftOf*) of one of the windows, or a value that is subtracted from below (*ImBelow*) or right (*ImRightOf*) of one of the windows.

See also: *TLayoutConstraint struct*

TReplaceDialog class

findrepl.h

TReplaceDialog creates a modeless dialog box that lets the user enter a selection of text to replace.

Public constructors**Constructor**

```
TReplaceDialog(TWindow* parent, TData& data, TResID templateName=0,
               const char far* title=0, TModule* module=0);
```

Constructs a *TReplaceDialog* object with a parent window, resource ID, and caption. Sets the attributes of the dialog box with the specified data.

See also: *TFindReplaceDialog::TData*

Protected member functions**DoCreate**

```
HWND DoCreate();
```

Creates a modeless dialog box.

See also: *TDialog::DoCreate*

TResId class

TResID is a simple support class that creates a resource ID object from either numerical or string resource identifier. This resource ID object can be passed to various ObjectWindows classes.

Public constructors

Constructor

```
TResID();
```

The default *TResID* constructor. Sets *Id* to 0.

Constructor

```
TResID(int resNum);
```

Creates a *TResID* object with the given *resNum*. Uses the Windows macro `MAKEINTRESOURCE` to set *Id* to a compatible resource string for Windows resource-management functions.

Constructor

```
TResID(LPCSTR resString);
```

Creates a *TResID* object with the given *resString*. *Id* is set to *resString*.

Public member functions

IsString

```
inline BOOL IsString() const;
```

Returns TRUE if this resource ID was created from a string; otherwise, returns FALSE.

operator LPSTR()

```
inline operator LPSTR();
```

Typecasting operator that converts *Id* to type LPSTR.

Friend functions

operator>>

```
friend ipstream& operator>>(ipstream& is, TResID& id);
```

Extracts a *TResID* object from *is* (the given input stream), and copies it to *id*. Returns a reference to the resulting stream, allowing the usual chaining of >> operations.

See also: *TResID* friend **operator<<**, *ipstream*

operator<<

```
friend opstream& operator<<(opstream& os, const TResID& id);
```

Inserts the given *TResID* object (*id*) into the *opstream* (*os*). Returns a reference to the resulting stream, allowing the usual chaining of << operations.

See also: *TResID* **friend operator>>**, *opstream*

operator<<

```
friend ostream& operator<<(ostream& os, const TResID& id);
```

Formats and inserts the given *TResID* object (*id*) into the *ostream* (*os*). Returns a reference to the resulting stream, allowing the usual chaining of << operations.

See also: *TResID* **friend operator>>**, *ostream*

TResponseTableEntry class

eventhan.h

A template class, *TResponseTableEntry* lets you define a pattern for entries into a response table. Entries consist of a message, a notification code, a resource ID, a dispatcher type, and a pointer to a member function. See Chapter 2 for a discussion of the format for response table entries.

See DECLARE_RESPONSE_TABLE and DEFINE_RESPONSE_TABLE for additional information about the macros in the response tables.

Public data members

Dispatcher

```
TAnyDispatcher Dispatcher;
```

An abstract dispatcher type, *Dispatcher* points to one of the dispatcher functions.

Id

```
UINT Id;
```

Contains the menu or accelerator resource ID (CM_xxxx) for the message response member function.

Msg

```
UINT Msg;
```

Contains the ID of the message sent. These can be command messages, child id messages, notify-based messages such as LBN_SELCHANGE, or Windows messages such as LBUTTONDOWN.

NotifyCode

```
UINT NotifyCode;
```

Stores the control notification code (for example, ID_LISTBOX) for the response table entry. These can be button, combo box, edit control, or list box notification codes.

Pmf

```
PMF Pmf;
```

Pmf points to the actual handler or member function.

T

```
typedef void(T::*PMF)();
```

Type for generic member function that responds to notification messages. *T* is the template for the response table.

TRgbQuad class**color.h**

TRgbQuad is a support class derived from the structure *tagRGBQUAD*, which is defined as follows:

```
typedef struct tagRGBQUAD {
    BYTE  rgbBlue;
    BYTE  rgbGreen;
    BYTE  rgbRed;
    BYTE  rgbReserved;
} RGBQUAD;
```

where *rgbBlue*, *rgbGreen*, and *rgbRed* specify the relative blue, green, and red intensities of a color. *rgbReserved* is not used and must be set to 0.

TRgbQuad is used in conjunction with the classes *TPalette* and *TColor* to simplify RGBQUAD-based color operations. Constructors are provided to create *TRgbQuad* objects from explicit RGB values, from *TColor* objects, or from other *TRgbQuad* objects.

Public constructors**Constructor**

```
TRgbQuad(int r, int g, int b);
```

Creates a *TRgbQuad* object with *rgbRed*, *rgbGreen*, and *rgbBlue* set to *r*, *g*, and *b* respectively. *rgbReserved* is set to 0.

See also: *tagRGBQUAD* struct

Constructor

```
TRgbQuad(TColor c);
```

Creates a *TRgbQuad* object with *rgbRed*, *rgbGreen*, and *rgbBlue* set to *c.Red()*, *c.Green()*, and *c.Blue()* respectively. *rgbReserved* is set to 0.

See also: *tagRGBQUAD* struct, *TColor::Red*, *TColor::Green*, *TColor::Blue*

Constructor

```
TRgbQuad(const RGBQUAD far& q);
```

Creates a *TRgbQuad* object with the same values as the referenced RGBQUAD object.

See also: struct *tagRGBQUAD*

TRgbTriple class

color.h

TRgbTriple is a support class derived from the structure *tagRgbTriple*, which is defined as follows:

```
typedef struct tagRGBTRIPLE {
    BYTE  rgbBlue;
    BYTE  rgbGreen;
    BYTE  rgbRed;
} RGBTRIPLE;
```

where *rgbBlue*, *rgbGreen*, and *rgbRed* specify the relative blue, green, and red intensities for a color.

TRgbTriple is used in conjunction with the classes *TPalette* and *TColor* to simplify bmci-color-based operations. Constructors are provided to create *TRgbTriple* objects from explicit RGB values, from *TColor* objects, or from other *TRgbTriple* objects.

Public constructors

Constructor

```
TRgbTriple(int r, int g, int b);
```

Creates a *TRgbTriple* object with *rgbRed*, *rgbGreen*, and *rgbBlue* set to *r*, *g*, and *b* respectively.

See also: *tagRGBTRIPLE* struct

Constructor

```
TRgbTriple(TColor c);
```

Creates a *TRgbTriple* object with *rgbRed*, *rgbGreen*, and *rgbBlue* set to *c.Red()*, *c.Green()*, and *c.Blue()* respectively. *rgbReserved* is set to 0.

See also: *tagRGBTRIPLE* struct, *TColor::Red*, *TColor::Green*, *TColor::Blue*

Constructor

```
TRgbTriple(const RGBTRIPLE far& t);
```

Creates a *TRgbTriple* object with the same values as the referenced RGBTRIPLE object.

See also: *tagRGBTRIPLE* struct

TScreenDC class

dc.h

Derived from *TWindowDC*, *TScreenDC* is a DC class that provides direct access to the screen bitmap. *TScreenDC* gets a DC for Window handle 0, which is for the whole screen with no clipping. Window handle 0 paints on top of other windows.

Public constructors

Constructor

```
TScreenDC();
```

Default constructor for *TScreenDC* objects.

TScrollBar

scrollba.h

TScrollBar objects represent standalone vertical and horizontal scroll bar controls. Most of *TScrollBar*'s member functions manage the scroll bar's sliding box (thumb) position and range.

One special feature of the type *TScrollBar* is the notify-based set of member functions that automatically adjust the scroll bar's thumb position in response to Windows scroll bar messages.



Never place *TScrollBar* objects in windows that have either the `WS_HSCROLL` or `WS_VSCROLL` styles in their attributes.

TScrollBar is a streamable class.

Public data members

LineMagnitude

```
int LineMagnitude;
```

LineMagnitude is the number of range units to scroll the scroll bar when the user requests a small movement by clicking on the scroll bar's arrows.

TScrollBar's constructor sets *LineMagnitude* to 1 by default. (The scroll range is 0-100 by default.)

See also: *TScrollBar::SetupWindow*

PageMagnitude

```
int PageMagnitude;
```

PageMagnitude is the number of range units to scroll the scroll bar when the user requests a large movement by clicking in the scroll bar's scrolling area.

TScrollBar's constructor sets *PageMagnitude* to 10 by default. (The scroll range is 0-100 by default.)

Public constructors

Constructor

```
TScrollBar(TWindow* parent, int id, int x, int y, int w, int h,
           BOOL isHScrollBar, TModule* module = 0);
```

Constructs and initializes a *TScrollBar* object with the given parent window (*parent*), a control ID (*id*), a position (*x, y*), and a size of (*w, h*). Invokes the *TControl* constructor with similar parameters. If *isHScrollBar* is TRUE, adds SBS_HORZ to the styles specified in *Attr.Style*. If not TRUE, adds SBS_VERT. If the supplied height for a horizontal scroll bar or the supplied width for a vertical scroll bar is 0, a standard value is used. *LineMagnitude* is initialized to 1 and *PageMagnitude* is set to 10.

See also: *TControl::TControl*

Constructor

```
TScrollBar(TWindow* parent, int resourceId, TModule* module = 0);
```

Constructs a *TScrollBar* object to be associated with a scroll bar control of a *TDialog*. Invokes the *TControl* constructor with identical parameters.

The *resourceId* parameter must correspond to a scroll bar resource that you define.

Public member functions

DeltaPos

```
virtual int DeltaPos(int delta);
```

Calls *SetPosition* to change the scroll bar's thumb position by the value supplied in *delta*. A positive *delta* moves the thumb down or right. A negative *delta* value moves the thumb up or left. The new thumb position is returned.

See also: *TScrollBar::SetPosition*

GetPosition

```
inline virtual int GetPosition() const;
```

Returns the scroll bar's current thumb position.

See also: *TScrollBar::SetPosition*, *TScrollBarData* struct

GetRange

```
inline virtual void GetRange(int& min, int& max) const;
```

Returns the end values of the present range of scroll bar thumb positions in *min* and *max*.

See also: *TScrollBar::SetPosition*, *TScrollBar::SetRange*, *TScrollBarData* struct

SBBottom

virtual void SBBottom();

Calls *SetPosition* to move the thumb to the bottom or right of the scroll bar. *SB_BOTTOM* is called to respond to the thumb being dragged to the bottom or rightmost position of the scroll bar.

See also: *TScrollBar::SetPosition*

SBLineDown

virtual void SBLineDown();

Calls *SetPosition* to move the thumb down or right (by *LineMagnitude* units). *SBLineDown* is called to respond to a click on the bottom or right arrow of the scroll bar.

See also: *TScrollBar::SetPosition*

SBLineUp

virtual void SBLineUp();

Calls *SetPosition* to move the thumb up or left (by *LineMagnitude* units). *SBLineUp* is called to respond to a click on the top or left arrow of the scroll bar.

See also: *TScrollBar::SetPosition*

SBPageDown

virtual void SBPageDown();

Calls *SetPosition* to move the thumb down or right (by *PageMagnitude* units). *SBPageDown* is called to respond to a click in the bottom or right scrolling area of the scroll bar.

See also: *TScrollBar::SetPosition*

SBPageUp

virtual void SBPageUp();

Calls *SetPosition* to move the thumb up or left (by *PageMagnitude* units). *SBPageUp* is called to respond to a click in the top or left scrolling area of the scroll bar.

See also: *TScrollBar::SetPosition*

SBThumbPosition

virtual void SBThumbPosition(UINT thumbPos);

Calls *SetPosition* to move the thumb. *SBThumbPosition* is called to respond when the thumb is set to a new position.

See also: *TScrollBar::SetPosition*

SBThumbTrack

virtual void SBThumbTrack(UINT thumbPos);

Calls *SetPosition* to move the thumb as it is being dragged to a new position.

See also: *TScrollBar::SetPosition*

- SBTop** virtual void SBTop();
Calls *SetPosition* to move the thumb to the top or right of the scroll bar. *SBTop* is called to respond to the thumb being dragged to the top or rightmost position on the scroll bar.
See also: *TScrollBar::SetPosition*
- SetPosition** virtual void SetPosition(int thumbPos);
Moves the thumb to the position specified in *ThumbPos*. If *ThumbPos* is outside the present range of the scroll bar, the thumb is moved to the closest position within range.
See also: *TScrollBar::GetPosition*
- SetRange** inline virtual void SetRange(int min, int max);
Sets the scroll bar to the range between *min* and *max*.
See also: *TScrollBar::GetRange*
- Transfer** virtual UINT Transfer(void* buffer, TTransferDirection direction);
Transfers scroll bar data to or from the transfer buffer pointed to by *buffer*. *buffer* is expected to point to a *TScrollBarData* structure.
Data is transferred to or from the transfer buffer if *tdGetData* or *tdSetData* is supplied as the *direction*.
Transfer always returns the size of the transfer data (the size of the *TScrollBarData* structure). To retrieve the size of this data without transferring data, pass *tdSizeData* as the *direction*.
See also: *TScrollBarData* struct

Protected member functions

- GetClassName** char far* GetClassName();
Returns the name of *TScrollBar*'s Windows registration class, "SCROLLBAR."
- SetupWindow** void SetupWindow();
Sets the scroll bar's range to 0, 100. To redefine this range, call *SetRange*.
See also: *TScrollBar::SetRange*

TScrollBarData struct**scrollba.h**

The *TScrollBarData* structure contains integer values that represent a range of thumb position on the scroll bar. *TScrollBar*'s function *GetRange* calls *TScrollBarData* to obtain the highest and lowest thumb positions on the scroll bar. *GetPosition* calls *TScrollBarData* to obtain the current thumb position on the scroll bar.

See also: *TScrollBar::Transfer*

Public data members**HighValue**

int HighValue;

Contains the highest value of the thumb position in the scroll bar's range.

See also: *TScrollBar::GetRange*

LowValue

int LowValue;

Contains the lowest value of thumb position in the scroll bar's range.

See also: *TScrollBar::GetRange*

Position

int Position;

Contains the scroll bar's thumb position.

See also: *TScrollBar::GetPosition*

TScroller class**scroller.h**

TScroller supports an automatic window-scrolling mechanism (referred to as autoscrolling) that works in conjunction with horizontal or vertical window scroll bars (it also works if there are no scroll bars). When autoscrolling is activated, the window automatically scrolls when the mouse is dragged from inside the client area of the window to outside that area. If the *AutoMode* data member is TRUE, *TScroller* performs autoscrolling.

To use *TScroller*, set the *Scroller* member of your *TWindow* descendant to a *TScroller* object instantiated in the constructor of your *TWindow* descendant. *TScroller* is a streamable class.

Public data members

AutoMode	<p>BOOL AutoMode;</p> <p>Is TRUE if automatic scrolling is activated.</p>
AutoOrg	<p>BOOL AutoOrg;</p> <p>Is TRUE if scroller offsets original.</p>
HasHScrollBar, HasVScrollBar	<p>BOOL HasHScrollBar, HasVScrollBar;</p> <p>Is TRUE if scroller has horizontal or vertical scroll.</p>
TrackMode	<p>BOOL TrackMode;</p> <p>Is TRUE if track scrolling is activated.</p>
Window	<p>TWindow* Window;</p> <p>Points to the window whose client area scroller is to be managed.</p>
XLine, YLine	<p>int XLine, YLine;</p> <p>Specifies the number of logical device units per line to scroll the rectangle in the horizontal (<i>X</i>) and vertical (<i>Y</i>) directions.</p>
XPage, YPage	<p>int XPage, YPage;</p> <p>Specifies the number of logical device units per page to scroll the rectangle in the horizontal (<i>X</i>) and vertical (<i>Y</i>) directions.</p>
XPos, YPos	<p>long XPos, YPos;</p> <p>Specifies the current position of the rectangle in horizontal (<i>XPos</i>) and vertical (<i>YPos</i>) scroll units.</p>
XRange, YRange	<p>long XRange, YRange;</p> <p>Specifies the number of horizontal and vertical scroll units.</p>
XUnit, YUnit	<p>int XUnit, YUnit;</p> <p>Specifies the amount (in logical device units) to scroll the rectangle in the horizontal (<i>X</i>) and vertical (<i>Y</i>) directions. The rectangle is scrolled right if <i>XUnit</i> is positive and left if <i>XUnit</i> is negative. The rectangle is scrolled down if <i>YUnit</i> is positive and up if <i>YUnit</i> is negative.</p>

Public constructors and destructor

Constructor	<p>TScroller(TWindow* window, int xUnit, int yUnit, long xRange, long yRange);</p>
--------------------	--

Constructs a *TScroller* object with *window* as the owner window, and *xUnit*, *yUnit*, *xRange*, and *yRange* as *xUnit*, *yUnit*, *xRange* and *yRange*, respectively. Initializes data members to default values. *HasHScrollBar* and *HasVScrollBar* are set according to the scroll bar attributes of the owner window.

Destructor

```
virtual ~TScroller();
```

Destructs a *TScroller* object. Sets owning window's *Scroller* number variable to 0.

Public member functions

AutoScroll

```
virtual void AutoScroll();
```

Scrolls the owner window's display in response to the mouse being dragged from inside to outside the window. The direction and the amount by which the display is scrolled depend on the current position of the mouse.

BeginView

```
virtual void BeginView(TDC& dc, TRect& rect);
```

If *TScroller::AutoOrg* is TRUE (default condition), *BeginView* automatically offsets the origin of the logical coordinates of the client area by XPos, YPos during a paint operation. If *AutoOrg* is FALSE (for example, when the scroller is larger than 32,767 units) you must set the offset manually.

EndView

```
virtual void EndView();
```

Updates the position of the owner window's scroll bars to be coordinated with the position of the *TScroller*.

HScroll

```
virtual void HScroll(UINT scrollEvent, int thumbPos);
```

Responds to the specified horizontal *scrollEvent* by calling *ScrollBy* or *ScrollTo*. The type of scroll event is identified by the corresponding Windows SB_ constants. *thumbPos* contains the current thumb position when the scroller is notified of SB_THUMBTRACK and SB_THUMBPOSITION scroll events.

IsAutoMode

```
virtual BOOL IsAutoMode();
```

IsAutoMode is TRUE if automatic scrolling is activated.

See also: *TScroller::AutoMode*

IsVisibleRect

```
inline BOOL IsVisibleRect(long x, long y, int xExt, int yExt);
```

Is TRUE if the rectangle (*x*, *y*, *xExt*, and *yExt*) is visible.

SetPageSize

```
virtual void SetPageSize();
```

Sets the *XPage* and *YPage* data members to the width and height (in *XUnits* and *YUnits*) of the owner window's client area.

See also: *TScroller::XPage*, *TScroller::YPage*, *TScroller::XUnit*, *TScroller::YUnit*

SetRange

```
virtual void SetRange(long xRange, long yRange);
```

Sets the *xRange* and *yRange* of the *TScroller* to the parameters specified. Then calls *SetSBarRange* to synchronize the range of the owner window's scroll bars.

See also: *TScroller::SetSBarRange*

SetSBarRange

```
virtual void SetSBarRange();
```

Sets the range of the owner window's scroll bars to match the range of the *TScroller*.

SetUnits

```
virtual void SetUnits(int xUnit, int yUnit);
```

Sets the *XUnit* and *YUnit* data members to *TheXUnit* and *TheYUnit*, respectively. Updates *XPage* and *YPage* by calling *SetPageSize*.

See also: *TScroller::XPage*, *TScroller::YPage*, *TScroller::XUnit*, *TScroller::YUnit*

ScrollBy

```
inline void ScrollBy(long dx, long dy);
```

Scrolls to a position calculated using the passed delta values (*dx* and *dy*). A positive delta position moves the thumb position down or right. A negative delta position moves the thumb up or left.

ScrollTo

```
virtual void ScrollTo(long x, long y);
```

Scrolls the rectangle to the position specified in *x* and *y*.

SetWindow

```
inline void SetWindow(TWindow* win);
```

Sets the owning window to *win*.

VScroll

```
virtual void VScroll(UINT scrollEvent, int thumbPos);
```

Responds to the specified vertical *scrollEvent* by calling *ScrollBy* or *ScrollTo*. The type of scroll event is identified by the corresponding Windows *SB_* constants. *thumbPos* contains the current thumb position when the scroller is notified of *SB_THUMBTRACK* and *SB_THUMBPOSITION* scroll events.

See also: *TScroller::ScrollTo*

XScrollValue

```
inline int XScrollValue(long rangeUnit);
```

XScrollValue converts a horizontal range value from the scroll bar to a horizontal scroll value.

See also: *TScroller::YScrollValue*

XRangeValue

```
inline int XRangeValue(int scrollUnit);
```

XRangeValue converts a horizontal scroll value from the scroll bar to a horizontal range value.

See also: *TScroller::YRangeValue*

YScrollValue

```
inline int YScrollValue(long rangeUnit);
```

YScrollValue converts a vertical range value from the scroll bar to a vertical scroll value.

See also: *TScroller::XScrollValue*

YRangeValue

```
inline int YRangeValue(int scrollUnit);
```

YRangeValue converts a vertical scroll value from the scroll bar to a vertical range value.

See also: *TScroller::XRangeValue*

TSeparatorGadget class**gadget.h**

TSeparatorGadget is a simple class you can use to create a separator between gadgets. To do so, you must specify the size of the separator in units of `SM_CXBORDER` (width of the window frame) and `SM_CYBORDER` (height of the window frame). The right and bottom boundaries of the separator are set after calling *GetSystemMetrics*. By default, the separator disables itself and turns off shrink-wrapping. Note that the default border style is none.

See also: *TGadget::TBorderStyle* enum

Public member functions**TSeparatorGadget**

```
TSeparatorGadget(int size = 6);
```

Used for both the width and the height of the separator, *size* is initialized at 6 border units (the width or height of a thin window border).

TSize is a support class derived from the structure *tagSIZE*. Under Win32, the latter is defined as

```
typedef struct tagSize {
    LONG x;
    LONG y;
} POINT;
```

Under Win16, *tagSize* is defined as

```
typedef struct
    tagSIZE {
        int x;
        int y;
    } POINT;

typedef struct tagSIZE {
    LONG cx;
    LONG cy;
} SIZE;
```

TSize encapsulates the notion of a two-dimensional quantity that usually represents a displacement or the height and width of a rectangle. *TSize* inherits the two data members *cx* and *cy* from *tagSIZE*. As with *TPoint*, *TSize* objects can be created from a pair of **ints**, a point of type **POINT**, another value of type **SIZE**, or from the low and high words of a **DWORD** value. Member functions and operators are provided for comparing, assigning, and manipulating sizes. Overloaded **<<** and **>>** operators are declared as friends of *TSize*, allowing chained insertion and extraction of *TSize* objects with streams.

Public constructors

Constructor

```
inline TSize();
```

The default *TSize* constructor.

Constructor

```
inline TSize(int dx, int dy);
```

Creates a *TSize* object with *cx = dx* and *cy = dy*.

Constructor

```
inline TSize(const POINT& point);
```

Creates a *TSize* object with *cx = point.x* and *cy = point.y*.

See also: *Point*

Constructor `inline TSize(const SIZE& size);`
 Creates a *TSize* object with $cx = size.cx$ and $cy = size.cy$.

See also: *Size* struct

Constructor `inline TSize(DWORD dw);`
 Creates a *TSize* object with $cx = LOWORD(dw)$ and $cy = HIWORD(dw)$.

Public member functions

Magnitude `inline int Magnitude() const;`
 Returns the length of the diagonal of the rectangle represented by this object. The value returned is an **int** approximation to the square root of $(cx^2 + cy^2)$.

operator+ `inline TSize operator+(const TSize& size) const;`
 Calculates an offset to this *TSize* object using the given *size* argument as the displacement. Returns the object $(cx + size.cx, cy + size.cy)$. This *TSize* object is not changed.

See also: *TSize::operator-*

operator- `inline TSize operator-(const TSize& size) const;`
`inline TSize operator-() const;`
 The first version calculates a negative offset to this *TSize* object using the given *size* argument as the displacement. Returns the point $(cx - size.cx, cy - size.cy)$. This object is not changed.

The second version returns the *TSize* object $(-cx, -cy)$. This object is not changed.

See also: *TSize::operator+*

operator== `inline BOOL operator==(const TSize& other) const;`
 Returns TRUE if this size object is equal to the *other TSize* object; otherwise returns FALSE.

See also: *TSize::operator!=*

operator!= `inline BOOL operator!=(const TSize& other) const;`
 Returns FALSE if this size object is equal to the *other TSize* object; otherwise returns TRUE.

See also: *TSize::operator==*

operator+= inline TSize& operator+=(const TSize& size) const;

Offsets this *TSize* object by the given *size* argument. This *TSize* object is changed to $(cx + size.cx, cy + size.cy)$. Returns a reference to this object.

See also: *TSize::operator-=-*

operator-=- inline TSize& operator-=(const TSize& size) const;

Negatively offsets this *TSize* object by the given *size* argument. This object is changed to $(cx - size.cx, cy - size.cy)$. Returns a reference to this object.

See also: *TSize::operator+=*

Friend functions

operator>> friend inline ipstream& operator>>(ipstream& is, TSize& s) const;

Extracts a *TSize* object from *is*, the given input stream, and copies it to *s*. Returns a reference to the resulting stream, allowing the usual chaining of >> operations.

See also: *TSize* **friend operator<<, ipstream**

operator<< friend inline opstream& operator<<(opstream& os, const TSize& s) const;

Inserts the given *TSize* object (*s*) into the *opstream* (*os*). Returns a reference to the resulting stream, allowing the usual chaining of << operations.

See also: *TSize* **friend operator>>, opstream**

operator<< friend inline ostream& operator<<(ostream& os, const TSize& s) const;

Formats and inserts the given *TSize* object (*s*) into the *ostream* (*os*). The format is " $(cx \times cy)$ ". Returns a reference to the resulting stream, allowing the usual chaining of << operations.

See also: *TSize* **friend operator>>, ostream**

TSlider class

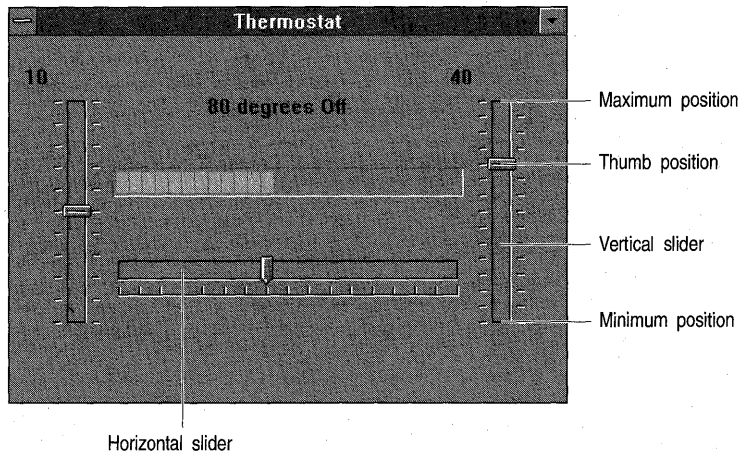
slider.h

An abstract base class derived from *TScrollBar*, *TSlider* defines the basic behavior of sliders (controls that are used for providing nonscrolling, position information). Like scroll bars, sliders have minimum and maximum positions as well as line and page magnitude. Sliders can be moved using either the mouse or the keyboard. If you use a mouse to move the slider, you can drag the thumb position, click on the slot on either side.

of the thumb position to move the thumb by a specified amount (*PageMagnitude*), or click on the ruler to position the thumb at a specific spot on the slider. The keyboard's Home and End keys move the thumb position to the minimum (*Min*) and maximum (*Max*) positions on the slider.

TSliders can cause the thumb positions to automatically align with the nearest tick positions (this is called snapping). You can also specify the tick gaps (the space between the lines that separate the major divisions of the X- or Y-axis).

The SLIDER.CPP ObjectWindows program on your distribution disk displays the following thermostat:



SLIDERAPP sets up the following constant values:

```
const WORD ID_THERMOSTAT = 201;
const WORD ID_HEATERTIME = 202;
const WORD ID_OUTSIDETEMP = 203;
const WORD ID_STATICTEMP = 205;
const WORD ID_STATICTIME = 206;
```

and then sets the following ruler ranges and positions:

```
TWindow::SetupWindow();
Thermostat->SetRange(40, 120);
Thermostat->SetRuler(5, FALSE);
Thermostat->SetPosition(75);

HeaterTime->SetRange(0, 20);
HeaterTime->SetRuler(2, FALSE);
HeaterTime->SetPosition(10);
```

```

OutsideTemp->SetRange(20, 90);
OutsideTemp->SetRuler(5, FALSE);
OutsideTemp->SetPosition(40);

Thermometer->SetRange(40-10, 120+10);
Thermometer->SetValue(75);
Thermometer->SetLed(4, 90);

```

before establishing the following values for static, gauge, and sliders:

```

SetTimer(ID_TIMER, 1000);
UpdateTemp();
UpdateHeaterTime();
UpdateOTemp();
StaticTemp = new TStatic(this, ID_STATICTEMP, "", 135, 40, 160, 17,0);
Thermometer = new TGauge(this, ID_THERMOMETER, 70, 90, 240, 20, TRUE, 2);
Thermostat = new THSlider(this, ID_THERMOSTAT, 70, 150, 240; 40);

```

For the complete program, see the `SLIDER.CPP` ObjectWindows program on your distribution disk.

See the two derived classes, *THSlider* and *TVSlider*, for specific details about horizontal and vertical sliders.

Public constructors and destructor

Constructor

```
TSlider(TWindow* parent, int id, int X, int Y, int W, int H,
        TResId thumbResId, TModule* module = 0);
```

Constructs a slider object setting *Pos* and *ThumbRgn* to 0, *TicGap* to *Range* divided by 10, *SlotThick* to 17, *Snap* to TRUE, and *Sliding* to FALSE. Sets *Attr.W* and *Attr.H* to the values in *X* and *Y*. *ThumbResId* is set to *thumbResId*.

Destructor

```
~TSlider();
```

Deconstructs a *TSlider* object and deletes *ThumbRgn*.

Public member functions

GetPosition

```
inline int GetPosition() const;
```

Returns the slider's current thumb position. Overloads *TScrollBar*'s virtual function.

See also: *TSlider::SetPosition*

GetRange

```
inline void GetRange(int &min, int &max) const;
```


Returns the end values of the present range of slider thumb positions in *min* and *max*. Overloads *TScrollBar*'s virtual function.

See also: *TSlider::SetRange*

SetPosition

```
void SetPosition(int thumbPos);
```

Moves the thumb to the position specified in *thumbPos*. If *thumbPos* is outside the present range of the slider, the thumb is moved to the closest position within the specified range. Overloads *TScrollBar*'s virtual function.

See also: *TSlider::GetPosition*

SetRange

```
void SetRange(int min, int max);
```

Sets the slider to the range between *min* and *max*. Overloads *TScrollBar*'s virtual function.

See also: *TSlider::GetRange*

SetRuler

```
inline void SetRuler(int ticGap, BOOL snap = FALSE);
```

Sets the slider's ruler. Each slider has a built-in ruler that is drawn with the slider. The ruler, which can be blank or have tick marks on it, can be created so that it forces the thumb to snap to the tick positions automatically.

Protected member functions

EvEraseBkgnd

```
BOOL EvEraseBkgnd(HDC hDC);
```

Responds to a *WM_ERASEBKGND* message and erases the background of the slider when the slider is changed. Calls the virtual functions *PaintRuler*, *PaintSlot*, and *PaintThumb* to paint the components of the slider. To avoid flickering, *EvEraseBkgnd* is called to erase the background as the painting occurs.

See also: *TSlider::EvPaint*

EvGetDlgCode

```
UINT EvGetDlgCode();
```

Responds to a *WM_GETDLGCODE* message and controls the response to a *DIRECTION* key or *TAB* key input. Captures cursor-movement keys to move the thumb by returning a *DLGC_WANTARROWS* message, which indicates that direction keys are desired.

EvKeyDown

```
void EvKeyDown(UINT key, UINT repeatCount, UINT flags);
```

EvKeyDown translates the virtual key code into a movement and then moves the thumb. *key* indicates the virtual key code of the pressed or key, *repeatCount* holds the number of times the same key is pressed, and *flags*

contains one of the following messages, which translate to virtual key (VK) codes:

Value	Virtual key code
SB_PAGEUP	VK_PRIOR
SB_PAGEDOWN	VK_NEXT
SB_BOTTOM	VK_END
SB_TOP	VK_HOME
SB_LINEUP	VK_LEFT(same as SB_LINELEFT)
SB_LINEUP	VK_UP
SB_LINEDOWN	VK_RIGHT(same as SB_LINERIGHT)
SB_LINEDOWN	VK_DOWN

EvKillFocus

```
void EvKillFocus(HWND hWndGetFocus);
```

In response to a WM_KILLFOCUS message sent to a window that is losing the keyboard, *EvKillFocus* hides and then destroys the caret.

EvLButtonDbtClk

```
void EvLButtonDbtClk(UINT modKeys, TPoint& point);
```

Responds to a WM_LBUTTONDOWNBLCLK message (which indicates the user double-clicked the left mouse button), then throws away the messages so the base class doesn't receive them.

EvLButtonDown

```
void EvLButtonDown(UINT modKeys, TPoint& point);
```

Responds to a mouse press by positioning the thumb or beginning a mouse drag. If the mouse is pressed down while it is over the thumb, *EvLButtonDown* enters sliding state. If the mouse is in the slot, *EvLButtonDown* pages up or down. If the mouse is on the ruler, *EvLButtonDown* jumps to that position. *EvLButtonDown* generates a scroll code of SB_THUMBPOSITION, SB_LINEUP, SB_LINEDOWN, SB_PAGEUP, SB_PAGEDOWN, SB_THUMBTRACK.

See also: *TSlider::EvLButtonUp*

EvLButtonUp

```
void EvLButtonUp(UINT modKeys, TPoint& point);
```

If the mouse button is released, *EvLButtonUp* ends sliding, paging, or jumping to a position on the ruler.

See also: *TSlider::EvLButtonDown*

EvMouseMove

```
void EvMouseMove(UINT modKeys, TPoint& point);
```

Moves the mouse to the indicated position. If the mouse is being dragged, *EvMouseMove* positions the thumb and sends the appropriate message to the parent window.

EvPaint

```
void EvPaint();
```

Paints the entire slider—ruler, slot, and thumb. Calls the virtual functions *PaintRuler*, *PaintSlot*, and *PaintThumb* to paint the components of the slider.

See also: *TSlider::EvEraseBkgnd*

EvSetFocus

```
void EvSetFocus(HWND hWndLostFocus);
```

Creates a blinking caret to show the focus in the current window.

See also: *TSlider::GetFocus*

EvSize

```
void EvSize(UINT sizeType, TSize& size);
```

Recalculates the size of the slider when the window size is changed.

GetBkColor

```
void GetBkColor(TDC& dc);
```

Sends a WM_CTLCOLOR message to the parent and calls *dc::GetBkColor* to extract the background color for the slider.

HitTest

```
virtual int HitTest(TPoint& point) = 0;
```

Gets information about where a given X, Y location falls on the slider. The return value is in *scrollCodes*. Each of the derived classes performs comparisons to return a scroll code.

See also: *TSlider::NotifyParent*

NotifyParent

```
virtual void NotifyParent(int scrollCode, int pos=0) = 0;
```

Sends a WS_HSCROLL or WS_VSCROLL message to the parent window.

See also: *THSlider::HitTest*, *TVSlider::HitTest*

PaintRuler

```
virtual void PaintRuler(TDC& dc) = 0;
```

Paints the ruler. It is assumed that the slot or thumb do not overlap the ruler.

PaintSlot

```
virtual void PaintSlot(TDC& dc) = 0;
```

Paints the slot in which the thumb slides.

PaintThumb

```
virtual void PaintThumb(TDC& dc);
```

Paint the thumb itself using a resource DIB translated to the current system button colors and which overlaps the slot.

PointToPos

```
virtual int PointToPos(TPoint& point) = 0;
```

Translates an X,Y point to a position in slider units.

See also: *TSlider::PosToPoint*

- PosToPoint** virtual TPoint PosToPoint(int pos) = 0;
 Translates a position in slider units to an X,Y point.
 See also: *TSlider::PointToPos*
- SetupThumbRgn** virtual void SetupThumbRgn();
 Creates the region that defines the thumb shape for this slider class. Although the default region is a simple bounding rectangle, it can be any shape. While the slider thumb is being moved, this region is used for testing the mouse position and updating the thumb position.
 See also: *ThumbRgn*
- SetupWindow** void SetupWindow();
 Calls *TScrollBar::SetupWindow* and *SetupThumbRgn* to set up the window.
 See also: *TScrollBar::SetupWindow*
- SlideThumb** virtual void SlideThumb(TDC& dc, int thumbPos);
 Slides the thumb to a given position and performs the necessary blitting and painting.
- SnapPos** int SnapPos(int pos);
 Constrains *Pos* so it is in the range from *Min* to *Max* and (if snapping is enabled) performs snapping by rounding *Pos* to the nearest *TicGap*.
 See also: *TSlider::TicGap*

Protected data members

- Bkcolor** TColor BkColor;
 Stores the background color of the slider.
- CaretRect** TRect CaretRect;
 Refers to the position of the caret's rectangle.
- Max** int Max;
 Contains the maximum value of the slider position.
- Min** int Min;
 Contains the minimum value of the slider position.
- MouseOffset** static TSize MouseOffset;
 Statics used while the mouse is down and the thumb is sliding.

Pos	int Pos; Indicates where the thumb is positioned on the slider. See also: <i>TSlider::GetPosition</i>
Range	UINT Range; Contains the difference between the maximum and minimum range of the slider.
SlideDC	static TDC* SlideDC; Statics used while the mouse is down and the thumb is sliding.
Sliding	BOOL Sliding; TRUE if the thumb is sliding.
SlotThick	int SlotThick; Indicates the thickness of the slot. Set to 17 by default.
Snap	BOOL Snap; TRUE if snapping is activated; otherwise FALSE.
ThumbRect	TRect ThumbRect; Holds the thumb's bounding rectangle.
ThumbResId	TResId ThumbResId; <i>ThumbResId</i> is the bitmap resource ID for the thumb knob.
ThumbRgn	TRegion* ThumbRgn; Refers to the region, if any, that defines the thumb shape for this slider class. See also: <i>TSlider::SetupThumbRgn</i>
TicGap	int TicGap; Specifies the amount of space in pixels between ticks.

Response table entries

Response table entry	Member function
EV_WM_ERASEBKGD	EvEraseBkgnd
EV_WM_GETDLGCODE	EvGetDlgCode
EV_WM_KEYDOWN	EvKeyDown

Response table entry	Member function
EV_WM_KILLFOCUS	EvKillFocus
EV_WM_LBUTTONDOWNCLK	EvLButtonDownClk
EV_WM_LBUTTONDOWN	EvLButtonDown
EV_WM_LBUTTONUP	EvLButtonUp
EV_WM_MOUSEMOVE	EvMouseMove
EV_WM_PAINT	EvPaint
EV_WM_SETFOCUS	EvSetFocus
EV_WM_SIZE	EvSize

TSortedStringArray typedef

validate.h

```
typedef TArrayAsVector<string> TSortedStringArray;
```

TSortedStringArray implements a sorted list of ASCII strings.

See also: *TValidator*, *TLookupValidator*

TStatic class

static.h

A *TStatic* is an interface object that represents a static text interface element in Windows. It must be used to create a static control in a parent *TWindow*. It can also be used to make it easier to modify the text of static controls in *TDialogs*.

Public data members

TextLen

```
UINT TextLen;
```

TextLen holds the size of the text buffer for static controls. The number of characters that can actually be stored in the static control is one less than *TextLen* because of the null terminator on the string. *TextLen* is also the number of bytes transferred by the *Transfer* member function.

Public constructors

Constructor

```
TStatic(TWindow* parent, int Id, const char far* title, int x, int y,
        int w, int h, UINT textLen, TModule* module = 0);
```

Constructs a static control object with the supplied parent window (*parent*), control ID (*Id*), text (*title*), position (*x, y*) relative to the origin of the parent

window's client area, width (*w*), height (*h*), and text length (*textLen*). By default, the static control is visible upon creation and has left-justified text. (*Attr.Style* is set to `WS_CHILD | WS_VISIBLE | WS_GROUP | SS_LEFT`.) Invokes a *TControl* constructor.

See also: *TControl::TControl*

Constructor

```
TStatic(TWindow* parent, int resourceId, UINT textLen,
        TModule* module = 0);
```

Constructs a *TStatic* object to be associated with a static control interface control of a *TDialog*. Invokes the *TControl* constructor with similar parameters, then sets *TextLen* to *textLen*. Disables the data transfer mechanism by calling *DisableTransfer*.

The *resourceId* parameter must correspond to a static control resource that you define.

See also: *TControl::TControl*

Public member functions

Clear

```
inline void Clear();
```

Clears the static control's text.

GetText

```
inline int GetText(char far* str, int maxChars);
```

Retrieves the static control's text, stores it in the *str* argument of *maxChars* size, and returns the number of characters copied.

GetTextLen

```
inline int GetTextLen();
```

Returns the length of the static control's text.

SetText

```
inline void SetText(const char far* string);
```

Sets the static control's text to the string supplied in *str*.

Transfer

```
inline virtual WORD Transfer(void* buffer, TTransferDirection direction);
```

Transfers *TextLen* characters of text to or from a transfer buffer pointed to by *buffer*. If *direction* is `tdGetData`, the text is transferred to the buffer from the static control. If *direction* is `tdSetData`, the static control's text is set to the text contained in the transfer buffer. *Transfer* returns *TextLen*, the number of bytes stored in or retrieved from the buffer. If *direction* is `tdSizeData`, *Transfer* returns *TextLen* without transferring data.

Protected member functions

GetClassName virtual char far* GetClassName();

Returns the name of *TStatic*'s Windows registration class (STATIC), or returns `STATIC_CLASS` if BWCC is enabled.

TStatus class

except.h

Used primarily for backward compatibility with previous versions of ObjectWindows, *TStatus* is used by *TModule* and *TWindow* to indicate an error in the initialization of an interface object. If *Status* is set to a nonzero value, a *TXCompatibility* exception is thrown.

Public constructors

Constructor TStatus();

Constructs a *TStatus* object and initializes the status code to 0.

See also: *TModule::Status*, *TWindow::Status*

Public data members

operator= inline TStatus& operator=(int statusCode);

Sets the status code and throws a *TXCompatibility* exception.

operator inline operator int() const;

Returns the status code.

TStatusBar class

statusba.h

In contrast to plain message bars, status bars provide several display options. ObjectWindows status bars let you include multiple text gadgets (the text on the left of the status bar) and different border styles. You can also reserve space for mode indicators (the text that displays the program's current state, such as extended selection (of keys and other modes), *CapsLock*, *NumLock*, *ScrollLock*, *Overwrite*, and macro recording). *TStatusBar* creates text gadgets for the mode indicators you request and adjusts the

spacing between mode indicators. The *TSpacing* struct stores spacing and layout unit constraints.

Like other control bars, the status bar is constructed and destroyed at the same time as its parent's window, but this is not a required procedure.

The following program statements show how to construct a status bar and insert it at the bottom of the window.

```
TStatusBar* sb = new TStatusBar(0, TGadget::Recessed,
    TStatusBar::CapsLock | TStatusBar::NumLock |
    TStatusBar::Overtyp);
frame->Insert(*sb, TDecoratedFrame::Bottom);

MainWindow = frame;
```

See the MDIFILE.CPP sample program on your distribution disk for an example of how to create a window with a status bar.

Public data members

TModeIndicator enum

```
enum TModeIndicator {ExtendSelection = 1, CapsLock = 1 << 1,
    NumLock = 1 << 2, ScrollLock = 1 << 3,
    Overtyp = 1 << 4, RecordingMacro = 1 << 5};
```

Enumerates the program modes. By default, these are arranged horizontally on the status bar from left to right.

Public constructors

Constructor

```
TStatusBar(TWindow* parent = 0,
    TGadget::TBorderStyle borderStyle = TGadget::Recessed,
    UINT modeIndicators = 0, TFont *font = new TGadgetWindowFont,
    TModule* module = 0);
```

Constructs a *TStatusBar* object in the *parent* window and creates any new gadgets and mode indicator gadgets. Sets *BorderStyle* to *borderStyle*, *ModeIndicators* to *modeIndicators*, and *NumModeIndicators* to 0. Sets the values of the margins and borders depending on whether the gadget is raised, recessed, or plain.

Public member functions

Insert

```
void Insert(TGadget& gadget, TPlacement = After, TGadget* sibling = 0);
```

Inserts the gadget in the status bar. By default, the new gadget is placed just after any existing gadgets and to the left of the status mode indicators.

operator `TTextGadget* operator[](UINT index);`

Returns a gadget at a given index, but cannot access mode indicator gadgets.

SetModeIndicator `void SetModeIndicator(TModeIndicator, BOOL state);`

Sets *TModeIndicator* to a given text gadget and set the status (on, by default) of the mode indicator. For the mode indicator to appear on the status bar, you must specify the mode when the window is constructed.

See also: *TStatusBar::TModeIndicator*, *TKeyboardModeTracker::EvKeyDown*

ToggleModeIndicator `void ToggleModeIndicator(TModeIndicator);`

Toggles the *ModeIndicator*.

See also: *TKeyboardModeTracker::EvKeyDown*

SetSpacing `inline void SetSpacing(TSpacing& spacing);`

Uses the *TSpacing* values to set the spacing to be used between mode indicator gadgets. *TSpacing* sets the status-bar margins in layout units. Typically, the message indicator (the leftmost text gadget) is left-justified on the status bar and the other indicators are right-justified. See *TLayoutMetrics* for an detailed explanation of layout units and constraints.

```
struct TSpacing {
    TMargins::TUnits Units;
    int Value;
    TSpacing() {Units = TMargins::LayoutUnits; Value = 0;}
};
```

See also: *TStatusBar::TModeIndicator*

Protected data members

BorderStyle `TGadget::TBorderStyle BorderStyle;`

One of the enumerated border styles—none, plain, raised, recessed, or embossed—used by the mode indicators on the status bar.

ModeIndicators `UINT ModeIndicators;`

The *ModeIndicators* bit field indicates which mode indicators have been created for the status bar.

NumModeIndicators `UINT NumModeIndicators;`

Specifies the number of mode indicators, which can range from 1 to 5.

Spacing

```
TSpacing Spacing;
```

Specifies the spacing between mode indicators on the status bar.

Protected member functions**PositionGadget**

```
void PositionGadget(TGadget* previous, TGadget* next, TPoint& point);
```

Determines the position of the new gadget in relation to any previously existing gadgets and uses the *Pixels*, *LayoutUnits*, and *BorderUnits* fields of *TMargins* to determine the amount of spacing to leave between the mode indicators.

TStream class**docview.h**

An abstract base class, *TStream* provides links between streams and documents, views, and document files.

Public data members**OpenMode**

```
int OpenMode;
```

Holds mode flags used when opening document streams. For example, the stream can be opened in *ofRead* mode to allow reading, but not changing (writing to) the file.

See also: *ofxxx document open* enum

StreamName

```
LPCSTR StreamName;
```

Holds the name of the stream used for opening the document.

Public destructor**Destructor**

```
inline ~TStream();
```

Closes the stream. Derived classes generally close the document if it was opened especially for this stream.

Public member functions

GetDocument `inline TDocument& GetDocument();`
 Returns the current document open for streaming.

Protected data members

Doc `TDocument& Doc;`
 Stores the document that owns this stream.

NextStream `TStream* NextStream;`
 Points to the next stream in the list of active streams.

Protected constructors

Constructor `TStream(TDocument& doc, LPCSTR name, int mode);`
 Constructs a *TStream* object. *doc* refers to the document object, *name* is the user-defined name of the stream, and *mode* is the mode used for opening the stream.
 See also: *TInStream*, *TOutStream*, *ofXXXX document open* enum, *shdocument sharing* enum

TStringLookupValidator class

validate.h

Derived from *TLookupValidator*, *TStringLookupValidator* is a streamable class. A *TStringLookupValidator* object verifies the data in its associated edit control by searching through a collection of valid strings. You can use string-lookup validators when your edit control needs to accept only members of a certain set of strings.

Public constructors and destructor

Constructor `TStringLookupValidator(TSortedStringArray strings);`
 Constructs a string-lookup object by first calling the constructor inherited from *TLookupValidator* and then setting *Strings* to *strings*.

Destructor `~TStringLookupValidator();`

Disposes of a list of valid strings by calling *NewStringList* and then disposes of the string-lookup validator object by calling the destructor inherited from *TLookupValidator*.

Public member functions

Error

```
void Error();
```

Overrides *TValidator*'s virtual function and displays a message box indicating that the typed string does not match an entry in the string list.

See also: *TValidator::Error*

Lookup

```
BOOL Lookup(const char far* str);
```

Overrides *TLookupValidator*'s virtual function. Returns TRUE if the string passed in *str* matches any of the *strings*. Uses the search method of the string collection to determine if *str* is present.

See also: *TLookupValidator::Lookup*

NewStringList

```
void NewStringList(TSortedStringArray strings);
```

Sets the list of valid input string for the string-lookup validator. Disposes of any existing string list and then sets *Strings* to *strings*.

Protected data member

Strings

```
TSortedStringArray Strings;
```

Points to a string collection containing all the valid strings the user can type. If *Strings* is NULL, all input is validated.

TSystemMenu class

menu.h

TSystemMenu creates a system menu object that then becomes the existing system menu. See Chapter 7 in the *ObjectWindows Programmer's Guide* for more information about menu objects.

Public constructors

Constructor

```
TSystemMenu(HWND wnd, BOOL revert = FALSE);
```

Constructs a system menu object. If *revert* is TRUE, then the menu created is a default system menu. Otherwise, it is the menu currently in the window.

See also: *TPopupMenu::TPopupMenu*

TTextGadget class

textgadg.h

Derived from *TGadget*, *TTextGadget* is a text gadget object. When you construct a text gadget, you must specify how many characters you want to reserve space for and how the text should be aligned horizontally. The inner boundaries of the text gadget are computed by multiplying the number of characters by the maximum character width.

Public data members

TAlign

```
enum TAlign {Left, Center, Right};
```

Enumerates the text-alignment attributes. *Left* aligns the text at the left edge of the bounding rectangle. *Right* aligns the text at the right edge of the bounding rectangle. *Center* aligns the text horizontally at the center of the bounding rectangle.

Public constructors

Constructor

```
TTextGadget(int id = 0, TBorderStyle = Recessed, TAlign = Left, UINT  
            numChars = 10, const char* text = 0);
```

Constructs a *TTextGadget* object with the specified ID, border style, and alignment. Sets *Margins.Left* and *Margins.Right* to 2. Sets *Text* and *TextLen* to 0.

Public member functions

GetText

```
inline char* GetText();
```

Returns the text for the gadget.

SetText

```
void SetText(const char* text);
```

If the text stored in *Text* is not the same as the new text, *SetText* deletes the text stored in *Text*. Then, it sets *TextLen* to the length of the new string. If no text exists, it sets both *Text* and *TextLen* to 0 and then calls *Invalidate* to invalidate the rectangle.

Protected data members

Align	TAlign Align; Text alignment attribute—left, center, or right-aligned.
NumChars	UINT NumChars; Holds the number of text characters.
Text	char* Text; Points to the text for the gadget.
TextLen	UINT TextLen; Stores the length of the text.

Protected member functions

GetDesiredSize	void GetDesiredSize(TSize &size); If shrink-wrapping is requested, <i>GetDesiredSize</i> returns the size needed to accommodate the borders, margins, and text; otherwise, if shrink-wrapping is not requested, it returns the gadget's current width and height. See also: <i>TGadget::GetDesiredSize</i>
Invalidate	void Invalidate(); Calls <i>TGadget::GetInnerRect</i> to compute the area of the text for the gadget and then <i>TGadget::InvalidateRect</i> to invalidate the rectangle in the parent window. See also: <i>TGadget::GetInnerRect</i> , <i>TGadget::Invalidate</i>
Paint	void Paint(TDC& dc); Calls <i>TGadget::PaintBorder</i> to paint the border. Calls <i>TGadget::GetInnerRect</i> to calculate the area of the text gadget's rectangle. If the text is left-aligned, <i>Paint</i> calls <i>dc.GetTextExtent</i> to compute the width and height of a line of the text. To set the background color, <i>Paint</i> calls <i>dc.GetSysColor</i> and sets the default background color to face shading (COLOR_BTNFACE). To set the

button text color, *Paint* calls *dc.SetTextColor* and sets the default button text color to `COLOR_BTNTEXT`. To draw the text, *Paint* calls *dc.ExtTextOut* and passes the parameters `ETO_CLIPPED` (so the text is clipped to fit the rectangle) and `ETO_OPAQUE` (so the rectangle is filled with the current background color).

See also: *TGadget::Paint*

TTileDirection enum

gadgetwi.h

Enumerates the horizontal and vertical direction for tiling the gadget.

```
enum TTileDirection;
```

See also: *TGadgetWindow::Direction*

TTinyCaption class

tinycapt.h

Derived from *TWindow*, *TTinyCaption* is a mix-in class that handles a set of non-client events to produce a smaller caption bar for a window. Whenever it displays the caption bar, *TTinyCaption* checks the window style and handles the Windows `WS_SYSMENU`, `WS_MINIMIZEBOX`, `WS_MAXIMIZEBOX` display attributes. Thus, you can use *TTinyCaption* to set the attributes of the tiny caption bar before enabling the caption. For example,

```
Attr.Style = WS_POPUP | WS_BORDER | WS_SYSMENU | WS_MINIMIZEBOX |  
WS_MAXIMIZEBOX;
```

TTinyCaption provides functions that let you manipulate Windows frame types, border styles, and menus. You can adjust the height of the caption bar or accept the default height, which is about one-half the height of a standard caption bar. If you set *CloseBox* to `TRUE`, then the window will close when you click the close box instead of displaying the system menu.

`OWL_CMD.CPP`, in your `OWL_APPS\OWL_CMD` directory, displays the following tiny caption bar:



If you are using *TTinyCaption* as a mix-in class that does partial event handling, call the *DoXxxx* function in the mix-in class (instead of the *EvXxxx* function) to avoid duplicating default processing. The following example from OWLCMD.CPP (a sample program on your distribution disk) illustrates this process:

```
void TMyFrame::EvSysCommand(UINT cmdType, TPoint& p)
{
    if (TTinyCaption::DoSysCommand(cmdType, p) == esPartial)
        TFrameWindow::EvSysCommand(cmdType, p);
}
```

The *TFloatingFrame* class can be used with *TTinyCaption* to produce a close box. See the sample programs OWLCMD.CPP and MDIFILE.CPP on your distribution disk for examples of how to use *TTinyCaption*.

Protected data members

Border	TSize Border; Thin frame border size for dividers.
CaptionHeight	int CaptionHeight; Height of the caption bar.
CaptionFont	TFont* CaptionFont; Font used for the text in the tiny caption bar.
CloseBox	BOOL CloseBox; If TRUE, the window will close when the close box is clicked.
DownHit	UINT DownHit; Location of mouse-button press or cursor move.
Frame	TSize Frame; Actual left and right, top and bottom dimensions of the caption bar.
isPressed	BOOL IsPressed; Is TRUE if a mouse button is pressed.
TCEnabled	BOOL TCEnabled; Is TRUE if the tiny caption bar is displayed.
WaitingForSysCmd	BOOL WaitingForSysCmd; Is TRUE if <i>TTinyCaption</i> is ready to receive system messages.

Protected constructors and destructor

Constructor

TTinyCaption():

Constructs a *TTinyCaption* object attached to the given parent window. Initializes the caption font to 0 and *TCEnabled* to FALSE so that the tiny caption bar is not displayed automatically.

Destructor

~TTinyCaption():

Destroys a *TTinyCaption* object and deletes the caption font.

Protected member functions

DoCommand

TEventStatus DoCommand(UINT id, HWND hWndCtl, UINT notifyCode, LRESULT& evRes);

Displays the system menu using *::TrackPopup* so that *TTinyCaption* sends WM_COMMAND instead of WM_SYSCOMMAND messages. If a system menu command is received, it's then transformed into a WM_SYSCOMMAND message. If the tiny caption bar is FALSE, *DoCommand* returns *esPartial*.

See also: *TTinyCaption::EvCommand*, *TEventStatus* enum

DoLButtonUp

TEventStatus DoLButtonUp(UINT hitTest, TPoint& screenPt);

Releases the mouse capture if the caption bar is enabled and a mouse button is pressed. Sets *hitTest*, indicating the mouse button has been pressed. Captures the mouse message and repaints the smaller buttons before returning *esComplete*.

See also: *TTinyCaption::EvLButtonUp*

DoMouseMove

TEventStatus DoMouseMove(UINT hitTest, TPoint& screenPt);

Returns *TEventStatus*.

DoNCActivate

TEventStatus DoNCActivate(BOOL active, BOOL& evRes);

If the tiny caption is not enabled or is iconic, returns *esPartial*. Otherwise, repaints the caption as an active caption and returns *esComplete*.

See also: *TTinyCaption::EvNCActivate*

DoNCCalcSize

TEventStatus DoNCCalcSize(BOOL calcValidRects, NCCALCSIZE_PARAMS far&calcSize, UINT& evRes);

If the caption bar is not enabled or is iconic, returns *esPartial*. Otherwise, calculates the dimensions of the caption and returns *esComplete*.

See also: *TTinyCaption::EvNCCalcSize*

DoNCHitTest

TEventStatus DoNCHitTest(TPoint& screenPt, UINT& evRes);

If the caption bar is not enabled, returns *esPartial*. Otherwise, sends a message to the caption bar that the mouse or the cursor has moved, and returns *esComplete*.

See also: *TTinyCaption::EvNCHitTest*

DoNCLButtonDown

TEventStatus DoNCLButtonDown(UINT hitTest, TPoint& screenPt);

If the caption bar isn't enabled, returns *esPartial*. Otherwise, determines if the user released the button outside or inside a menu, and returns *esComplete*.

See also: *TTinyCaption::EvNCLButtonDown*

DoNCPaint

TEventStatus DoNCPaint();

If the caption bar isn't enabled or is iconized, returns *esPartial*. Otherwise, gets the focus, paints the caption, and returns *esPartial*, thus indicating that a separate paint function must be called to paint the borders of the caption.

See also: *TTinyCaption::EvNCPaint*

DoSysCommand

TEventStatus DoSysCommand(UINT cmdType, TPoint& p);

If the caption bar isn't enabled, returns *esPartial*. If the caption bar is iconized and the user clicks the icon, calls *DoSysMenu* to display the menu in its normal mode and returns *esComplete*.

See also: *TTinyCaption::EvSysCommand*

DoSysMenu

void DoSysMenu();

Returns a handle to the system menu and makes a copy of the system menu.

EnableTinyCaption

void EnableTinyCaption(int ch=45, BOOL closeBox=FALSE);

Activates the tiny caption bar. By default, *EnableTinyCaption* replaces the system window with a tiny caption window that doesn't close when the system window is clicked. If the *closeBox* argument is TRUE, clicking on the system menu will close the window instead of bringing up the menu. You can use *EnableTinyCaption* to hide the window if you are using a tiny caption in a derived class. To diminish the tiny caption bar, try the following values:

EnableTinyCaption(30, TRUE);

Or, to maximize the tiny caption bar, use these values:

```
EnableTinyCaption(48, TRUE);
```

EvCommand

```
LRESULT EvCommand(UINT id, HWND hWndCtl, UINT notifyCode);
```

EvCommand provides extra processing for commands, but lets the focus window and its parent windows handle the command first.

See also: *TTinyCaption::DoCommand*

EvLButtonUp

```
void EvLButtonUp(UINT hitTest, TPoint& screenPt);
```

Responds to a mouse button-up message by calling *DoLButtonUp*. If *DoLButtonUp* doesn't return *IsComplete*, *EvLButtonUp* calls *TWindow::EvLButtonUp*.

See also: *TTinyCaption::DoLButtonUp*

EvMouseMove

```
void EvMouseMove(UINT hitTest, TPoint& screenPt);
```

Responds to a mouse-move message by calling *DoMouseMove*. If *DoMouseMove* doesn't return *IsComplete*, *EvMouseMove* calls *TWindow::EvMouseMove*.

See also: *TTinyCaption::DoMouseMove*

EvNCActivate

```
BOOL EvNCActivate(BOOL active);
```

Responds to a request to change a title bar or icon by calling *DoNCActivate*. If *DoNCActivate* doesn't return *esComplete*, *EvNCActivate* calls *TWindow::EvNCActivate*.

See also: *TTinyCaption::DoNCActivate*

EvNCCalcSize

```
UINT EvNCCalcSize(BOOL calcValidRects, NCCALCSIZE_PARAMS far& calcSize);
```

Responds to a request to change a title bar or icon by calling *DoNCActivate*. If *DoNCActivate* doesn't return *esComplete*, *EvNCActivate* calls *TWindow::EvNCActivate*.

Calculates the size of the command window including the caption and border so that it can fit within the window.

See also: *TTinyCaption::DoNCActivate*

EvNCHitTest

```
UINT EvNCHitTest(TPoint& screenPt);
```

Responds to a cursor move or press of a mouse button by calling *DoNCHitTest*. If *DoNCHitTest* doesn't return *esComplete*, *EvNCHitTest* calls *TWindow::EvNCHitTest*.

See also: *TTinyCaption::DoNCHitTest*

EvNCLButtonDown

```
void EvNCLButtonDown(UINT hitTest, TPoint& screenPt);
```

Responds to a press of the left mouse button while the cursor is within the nonclient area of the caption bar by calling *DoNCLButtonDown*. If *DoNCLButtonDown* doesn't return *esComplete*, *EvNCLButtonDown* calls *TWindow::EvNCLButtonDown*.

See also: *TTinyCaption::DoNCLButtonDown*

EvNCPaint

```
void EvNCPaint();
```

Responds to a request to change a title bar or icon by calling *DoNCActivate*. If *DoNCActivate* doesn't return *esComplete*, *EvNCActivate* calls *TWindow::EvNCActivate*.

Calls *TWindow::EvNCPaint* to paint the indicated device context or display screen.

See also: *TTinyCaption::DoNCActivate*

EvSysCommand

```
void EvSysCommand(UINT cmdType, TPoint& p);
```

Responds to a WM_SYSCOMMAND message by calling *DoSysCommand*. If *DoSysCommand* returns *esPartial*, *EvSysCommand* calls *TWindow::EvSysCommand*.

See also: *TTinyCaption::DoSysCommand*

GetCaptionRect

```
TRect GetCaptionRect();
```

Gets the area of the caption for changing or repainting.

See also: *TTinyCap::PaintCaption*

GetMaxBoxRect

```
TRect GetMaxBoxRect();
```

Returns the size of the maximize box rectangle.

See also: *TTinyCap::PaintMaxBoxRect*

GetMinBoxRect

```
TRect GetMinBoxRect();
```

Returns the size of the minimize box rectangle.

See also: *TTinyCap::PaintMinBoxRect*

GetSysBoxRect

```
TRect GetSysBoxRect();
```

Returns the size of the system box rectangle.

See also: *TTinyCap::PaintSysBoxRect*

PaintButton

```
void PaintButton(TDC& dc, TRect& boxRect, BOOL pressed);
```

Paints a blank button.

PaintCaption

```
void PaintCaption(TWindowDC &);
```

Calls *dc.SelectObject* to select the given rectangle and *dc.PatBlt* to paint the tiny caption bar using the currently selected brush for this device context.

See also: *TDC::SelectObject*, *TDC::PatBlt*

PaintCloseBox

```
void PaintCloseBox(TDC& dc, TRect& boxRect, BOOL pressed);
```

Paints a close box on the tiny caption bar. You can override the default box if you want to design your own close box.

See also: *TTinyCap::GetSysBoxRect*

PaintMaxBox

```
void PaintMaxBox(TDC& dc, TRect& boxRect, BOOL pressed);
```

Paints a maximize box on the tiny caption bar.

See also: *TTinyCap::GetMaxBoxRect*

PaintMinBox

```
void PaintMinBox(TDC& dc, TRect& boxRect, BOOL pressed);
```

Paints a minimize box on the tiny caption bar.

See also: *TTinyCap::GetMinBoxRect*

PaintSysBox

```
void PaintSysBox(TDC& dc, TRect& boxRect, BOOL pressed);
```

Paints the system box

See also: *TTinyCap::GetSysBoxRect*

Response table entries

Response table entry	Member function
EV_WM_NCACTIVATE	EvNcActivate
EV_WM_NCCALCSIZE	EvNcCalcSize
EV_WM_NCHITTEST	EvNcHitTest
EV_WM_NCPAINT	EvNcPaint
EV_WM_NCLBUTTONDOWN	EvNclButtonDown
EV_WM_LBUTTONDOWN	EvLButtonDown
EV_WM_MOUSEMOVE	EvMouseMove
EV_WM_SYSCOMMAND	EvSysCommand

TToolBox class**toolbox.h**

Derived from *TGadgetWindow*, *TToolBox* arranges gadgets in a matrix in which all columns are the same width (as wide as the widest gadget) and all rows are the same height (as high as the highest gadget).

You can specify exactly how many rows and columns you want for your toolbox, or you can let *TToolbox* calculate the number of columns and rows you need. If you specify *AS_MANY_AS_NEEDED*, the *TToolbox* calculates how many rows or columns are needed based on the opposite dimension. For example, if there are twenty gadgets, and you requested four columns, your matrix would have five rows.

Public constructors

Constructor

```
TToolBox(TWindow* parent, int numColumns = 2,
         int numRows = AS_MANY_AS_NEEDED,
         TTileDirection direction = Horizontal, TModule* module = 0);
```

Constructs a *TToolBox* object with the specified number of columns and rows and tiling direction. Overlaps the borders of the toolbox with those of the gadget and sets *ShrinkWrapWidth* to TRUE.

Public member functions

GetDesiredSize

```
void GetDesiredSize(TSize& size);
```

Overrides *TGadget's GetDesiredSize* function and computes the size of the cell by calling *GetMargins* to get the margins.

See also: *TGadgetWindow::GetDesiredSize*

Insert

```
void Insert(TGadget& gadget, TPlacement = After, TGadget* sibling = 0);
```

Overrides *TGadget's Insert* function and tells the button not to notch its corners.

See also: *TGadgetWindow::Insert*

SetDirection

```
virtual void SetDirection(TTileDirection direction);
```

Sets the direction of the tiling—either horizontal or vertical.

Protected data members

NumColumns

```
int NumColumns;
```

Contains the number of columns for the toolbox.

NumRows

```
int NumRows;
```

Contains the number of rows for the toolbox.

Protected member functions

TileGadgets

`TRect TileGadgets();`

Tiles the gadgets in the direction requested (horizontal or vertical). Calls *PositionGadget* to give derived classes an opportunity to adjust the spacing between gadgets.

See also: *TGadgetWindow::TileGadget*

TTransferDirection enum

window.h

TTransferDirection enum describes the following constants, which the transfer functions uses to determine how to transfer data to and from the transfer buffer.

Table 1.29
Transfer function
constants

Constant	Meaning
<code>tdGetData</code>	Retrieve data from the class.
<code>tdSetData</code>	Send data to the class.
<code>tdSizeData</code>	Return the size of data transferred by the class.

See also: *TWindow::Transfer*, *TWindow::TransferData*

TValidator class

validate.h

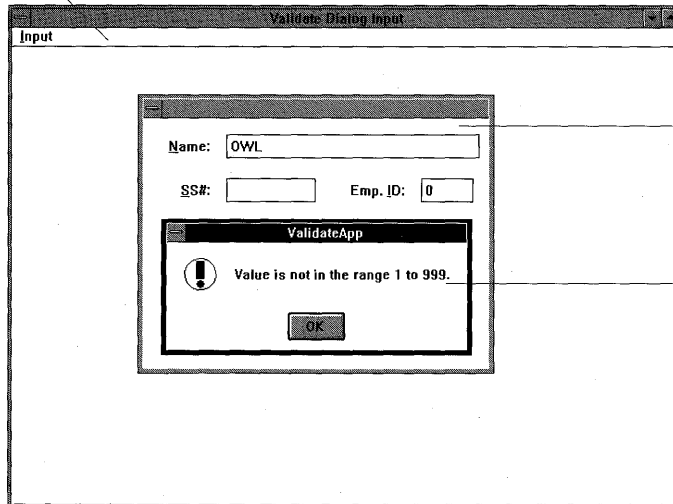
A streamable class, *TValidator* defines an abstract data validation object. Although you will never actually create an instance of *TValidator*, it provides the abstract functions for the other data validation objects.

The `VALIDATE.CPP` sample program on your distribution disk derives *TValidateApp* from *TApplication* in the following manner:

```
class TValidateApp : public TApplication {
public:
    TValidateApp() : TApplication("ValidateApp") {}
    void InitMainWindow() {
        MainWindow = new TTestWindow(0, "Validate Dialog Input");
    }
}
```


and displays the following message box if the user enter an invalid employee ID:

Main application window



After you choose InputEmployee from the menu, the Employee Data Entry dialog box appears.

If an invalid employee ID is entered, the ValidateApp message box appears.

Public constructors and destructor

Constructor

```
TValidator();
```

Constructs an abstract validator object and sets *Options* fields to 0.

Destructor

```
virtual ~TValidator() {}
```

Destroys an abstract validator object.

Public member functions

Error

```
virtual void Error();
```

Error is an abstract function called by *Valid* when it detects that the user has entered invalid information. By default, *TValidator::Error* does nothing, but derived classes can override *Error* to provide feedback to the user.

HasOption

```
inline BOOL HasOption(int option);
```

Gets the *Options* bits. Returns TRUE if a specified option is set.

See also: *TValidator::Options*, *Vxxxxx* constants

IsValid

```
virtual BOOL IsValid(const char far* str);
```

By default, *TValidator::IsValid* returns TRUE. Derived validator types can override *IsValid* to validate data for a completed edit control. If an edit control has an associated validator object, its *Valid* method calls the validator object's *Valid* method, which in turn calls *IsValid* to determine whether the contents of the edit control are valid.

See also: *TValidator::Valid*

IsValidInput

```
virtual BOOL IsValidInput(char far* str, BOOL suppressFill);
```

If an edit control has an associated validator object, it calls *IsValidInput* after processing each keyboard event. This gives validators such as filter validators an opportunity to catch errors before the user fills the entire item or screen.

By default, *IsValidInput* returns TRUE. Derived data validators can override *IsValidInput* to validate data as the user types it, returning TRUE if *str* holds valid data and FALSE otherwise.

str is the current input string. *suppressFill* determines whether the validator should automatically format the string before validating it. If *suppressFill* is TRUE, validation takes place on the unmodified string *str*. If *suppressFill* is FALSE, the validator should apply any filling or padding before validating data. Of the standard validator objects, only *TPXPictureValidator* checks *suppressFill*.

IsValidInput can modify the contents of the input string; for example, it can force characters to uppercase or insert literal characters from a format picture. *IsValidInput* should not, however, delete invalid characters from the string. By returning FALSE, *IsValidInput* indicates that the edit control should erase the incorrect characters.

SetOption

```
inline void SetOption(int option);
```

Sets the bits for the *Options* data member.

See also: *TValidator::Options*, *Voxxxx* constants

Transfer

```
virtual UINT Transfer(char far* str, void* buffer,  
                    TTransferDirection direction);
```

Allows a validator to set and read the values of its associated edit control. This is primarily useful for validators that check non-string data, such as numeric values. For example, *TRangeValidator* uses *Transfer* to read and write values instead of transferring an entire string.

By default, edit controls with validators give the validator the first chance to respond to *DataSize*, *GetData*, and *SetData* by calling the validator's *Transfer* method. If *Transfer* returns anything other than 0, it indicates to the edit control that it has handled the appropriate transfer. The default action of *TValidator::Transfer* is to always return 0. If you want the validator to transfer data, you must override its *Transfer* method.

Transfer's first two parameters are the associated edit control's text string and the *tdGetData* or *tdSetData* data record. Depending on the value of *direction*, *Transfer* can set *str* from *buffer* or read the data from *str* into *buffer*. The return value is always the number of bytes transferred.

If *direction* is *tdSizeData*, *Transfer* doesn't change either *str* or *buffer*; it just returns the data size. If *direction* is *tdSetData*, *Transfer* reads the appropriate number of bytes from *buffer*, converts them into the proper string form, and sets them into *str*, returning the number of bytes read. If *direction* is *tdGetData*, *Transfer* converts *str* into the appropriate data type and writes the value into *buffer*, returning the number of bytes written.

See also: *TTransferDirection* enum

UnsetOption

```
inline void UnsetOption(int option);
```

Unsets the bits specified in the *Options* data member.

See also: *TValidator::Options*, *Vxxxx* constants

Valid

```
inline BOOL Valid(const char far* str);
```

Returns TRUE if *IsValid* returns TRUE. Otherwise, calls *Error* and returns FALSE. A validator's *Valid* method is called by the *Valid* method of its associated edit control.

Edit controls with associated validator objects call the validator's *Valid* method under two conditions. The first condition is when the edit control's *ofValidate* option is set and the edit control calls *Valid* when it loses focus. The second condition is when the dialog box that contains the edit control calls *Valid* for all its controls, usually because the user requested to close the dialog box or to accept an entry screen.

Protected data members

Options

```
WORD Options;
```

Options is a bitmap member used to control options for various descendants of *TValidator*. By default, the *TValidator* constructor clears all the bits in *Options*.

See also: *voxxxx* constants, *TValidator::SetOption*, *TValidator::UnsetOption*

TValidator::TXValidator class

validate.h

A nested class, *TXValidator* describes an exception that results from an invalid validator object.

Public constructors

Constructor

```
TXValidator(UINT resId = IDS_VALIDATORSYNTAX );
```

Constructs a *TXValidator* object, setting the resource ID to *IDS_VALIDATORSYNTAX*.

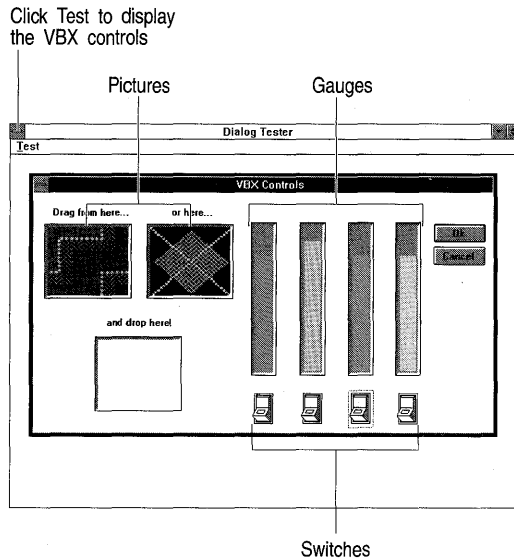
TVbxControl class

vbxctl.h

Derived from *TControl*, *TVbxControl* provides the interface for Visual Basic (VBX) controls. You can use this class to get or set the properties of VBX controls. Under certain conditions, you can also use additional methods for processing controls.

You can manipulate the control's properties using either an index value or a name. Several overloaded *GetProp* functions are provided so that you can access different types of properties. Similarly, several overloaded *SetProp* functions let you set the properties of controls using either the name of the VBX control or the index value. Consult the documentation for your VBX controls to find the name that corresponds to the property you want to manipulate.

The *VBXCTLX.CPP* sample program on your distribution disks displays the following VBX controls:



If you want the TVbx control object to process VBX events, you must derive a class from *TVbxControl* and add a response table that has entries for the events you want processed. For information about creating response tables and handling VBX messages, see *TVbxEventHandler*.

For more information about how to design programs that use VBX controls, see Chapter 15 in the *ObjectWindows Programmer's Guide*.

Public constructors and destructor

Constructor

```
TVbxControl(TWindow* parent, int id, const char far* vbxName, const char
            far* vbxClass, const char far* title, int x, int y, int w, int
            h, TModule* module = 0);
```

Constructs a VBX control where *parent* points to the parent window, *id* is the control's ID, *vbxName* is the name of the file containing the VBX control, *vbxClass* is the VBX class name, *title* is the control's caption, *x* and *y* are the coordinates in the parent window where the controls are to be placed, *w* and *h* are the width and height of the control, and *module* is the library resource ID for the control.

Constructor

```
TVbxControl(TWindow* parent, int resourceId, TModule* module = 0);
```

If a VBX control is part of a dialog resource, its ID can be used to construct a corresponding (or alias) ObjectWindows object. You can use this constructor if a VBX control has already been defined in the application's

resource file. *resourceId* is the resource ID of the VBX control in the resource file.

Destructor

```
~TVbxControl();
```

Destroys the *TVbxControl* object.

Public member functions

AddItem

```
inline BOOL AddItem(int index, const char far* item);
```

Adds an item (*item*) to the list of VBX control items at the specified index (*index*). Returns nonzero if successful.

Drag

```
BOOL Drag(int action);
```

Controls the drag and drop state of the VBX control according to the value of *action*, which can be 0 (cancel a drag operation), 1 (begin dragging a control), or 2 (end dragging a control).

GetEventIndex

```
inline int GetEventIndex(const char far* name);
```

Returns the index of the event associated with the name of the event passed in *name*. Returns -1 if an error occurs.

GetEventName

```
inline const char far* GetEventName(int eventindex);
```

Returns a string containing the name of an event associated with the integer event index number (*eventindex*). Returns 0 if an error occurs.

GetHCTL

```
HCTL GetHCTL();
```

Returns a handle to a VBX control associated with this *TVbxControl* object.

GetNumEvents

```
inline int GetNumEvents();
```

Returns the total number of events associated with the VBX control.

GetNumProps

```
inline int GetNumProps();
```

Returns the total number of properties associated with the VBX control.

GetProp

```
inline BOOL GetProp(int propIndex, int& value, int arrayIndex = -1);
```

Gets an integer property value. An overloaded function, *GetProp* gets a VBX control property using an index value. *propIndex* is the index value of the property whose value you want to get. *value* is a reference to the variable that will receive the property values. *arrayIndex*, an optional argument, specifies the position of the value in an array property if the property is an array type. If the property isn't an array type, *arrayIndex* defaults to -1. See the third-party reference guide for your VBX controls to

determine a property's data type. *GetProp* returns nonzero if successful. To get the property by specifying the property index, use one of the following six *GetProp* functions.

GetProp `inline BOOL GetProp(int propIndex long& value, int arrayIndex = -1);`

Gets a **long** property value.

GetProp `BOOL GetProp(int propIndex, ENUM& value, int arrayIndex=-1);`

Gets an enumerated property value. For example, a list of options associated with a font style might be defined as an enumerated type.

GetProp `BOOL GetProp(int propIndex, HPIC& value, int arrayIndex=-1);`

Gets a picture (*value*). *HPIC* is a handle to the picture.

GetProp `inline BOOL GetProp(int propIndex float& value, int arrayIndex = -1);`

Gets a floating-point property value.

GetProp `inline BOOL GetProp(int propIndex string& value, int arrayIndex = -1);`

Gets a string property value.

See also: *TVbxControl::SetProp*

GetProp `inline BOOL GetProp(const char far* name, int& value,
int arrayIndex = -1);`

Returns an integer data value. An overloaded function, *GetProp* gets a VBX control property. *propIndex* is the index value of the property whose value you want to get. *value* is a reference to the variable that will receive the property value. *arrayIndex*, an optional argument, specifies the position of the value in an array property if the property is an array type. If the property isn't an array type, *arrayIndex* defaults to -1. *GetProp* returns nonzero if successful. To get the property by specifying the name of the property, use one of the following five *GetProp* functions.

GetProp `inline BOOL GetProp(const char far* name, long& value,
int arrayIndex = -1);`

Gets a **long** property value.

GetProp `inline BOOL GetProp(const char far* name, float& value,
int arrayIndex = -1);`

Gets a floating-point property value.

Getprop `BOOL Getprop(const char far* name, ENUM& value, int arrayIndex=-1);`

Gets an enumerated property value.

Getprop `BOOL Getprop(const char far* name, HPIC& value, int arrayIndex=-1);`

Gets a picture (*value*) property value. *HPIC* is a handle to the picture.

GetProp

```
inline BOOL GetProp(const char far* name, string& value,
                   int arrayIndex = -1);
```

Gets a string property value.

See also: *TVbxControl::SetProp*

GetPropIndex

```
inline int GetPropIndex(const char far* name);
```

Gets the integer index value for the property name passed in *name*. Returns -1 if an error occurs. This usually indicates that the property name passed in *name* couldn't be located.

GetPropName

```
const char far* GetPropName(int index);
```

Gets the name for the property index passed in *index*. Returns 0 if an error occurs.

GetPropType

```
inline int GetPropType(int index);
```

Gets the type for the property specified by *index*. Returns 0 if an error occurs. The following table lists the names of the property types and their corresponding C++ data types.

Table 1.30
Property and C++
types

Property type	C++ type
PTYPE_CSTRING	HSZ
PTYPE_SHORT	short
PTYPE_LONG	LONG
PTYPE_BOOL	BOOL
PTYPE_COLOR	DWORD or COLORREF
PTYPE_ENUM	BYTE or ENUM
PTYPE_REAL	float
PTYPE_XPOS	LONG (Twips)
PTYPE_XSIZE	LONG (Twips)
PTYPE_YPOS	LONG (Twips)
PTYPE_YSIZE	LONG (Twips)
PTYPE_PICTURE	HPIC
PTYPE_BSTRING	HLSTR

IsArrayProp

```
inline BOOL IsArrayProp(int index);
```

Returns TRUE if the property specified by *index* is an array property.

Method

```
inline BOOL Method(int method, long far* args);
```

Used for invoking customized methods, *Method* returns TRUE if a VBX control can respond to the specified method (*method*).

Move

```
inline BOOL Move(int x, int y, int w, int h);
```


Moves a VBX control to the coordinates specified in *x* and *y*, which designate the upper left corner screen coordinates. Resizes the VBX control to *w* pixels wide by *h* pixels high. Returns nonzero if successful.

Refresh

```
inline BOOL Refresh();
```

Repaints the control's display area.

RemoveItem

```
inline BOOL RemoveItem(int index);
```

Removes an item (specified by *index*). The item could be removed from a list box, a combo box, or a database, for example.

SetProp

```
inline BOOL SetProp(int propIndex, int value, int arrayIndex = -1);
```

Sets the property to an integer value. An overloaded function, *SetProp* sets a VBX control property. *propIndex* is the index number of the property whose value you want to set. *value* specifies the new value for the property. *arrayIndex* specifies the position of the value in an array property if the property is an array type. If the property isn't an array type, *arrayIndex* defaults to -1. To set the property by passing the property's index value, use one of the following six *SetProp* functions.

SetProp

```
inline BOOL SetProp(int propIndex, long value, int arrayIndex = -1);
```

Sets the property to a **long** value.

SetProp

```
BOOL SetProp(int propIndex, ENUM value, int arrayIndex=-1);
```

Sets the property to an enumerated value.

SetProp

```
BOOL SetProp(int propIndex, HPIC value, int arrayIndex=-1);
```

Sets a picture to an HPIC, or picture, value.

SetProp

```
inline BOOL SetProp(int propIndex, float value, int arrayIndex = -1);
```

Sets the property to a floating-point value.

SetProp

```
inline BOOL SetProp(int propIndex, const string& value,  
int arrayIndex = -1);
```

Sets the property to a string value.

SetProp

```
inline BOOL SetProp(int propIndex, const char far* value,  
int arrayIndex = -1);
```

Sets the property to a character string value.

See also: *TVbxControl::GetProp*

SetProp

```
inline BOOL SetProp(const char far* name, int value, int arrayIndex = -1);
```

Sets the property to an integer value. An overloaded function, *SetProp* sets a VBX control property. *arrayIndex* specifies the position of the value in an array property if the property is an array type. If the property isn't an array type, *arrayIndex* defaults to -1. To set the property by using the property's name, use one of the following six *SetProp* functions.

SetProp `inline BOOL SetProp(const char far* name, long value,
int arrayIndex = -1);`

Sets the property to a **long** value.

SetProp `BOOL SetProp(int propIndex, ENUM value, int arrayIndex=-1);`

Sets the property to an enumerated value.

SetProp `BOOL SetProp(int propIndex, HPIC value, int arrayIndex = -1);`

Sets the picture property to an HPIC, or picture, value.

SetProp `inline BOOL SetProp(const char far* name, float value,
int arrayIndex = -1);`

Sets the property to a floating-point value.

SetProp `inline BOOL SetProp(const char far* name, const string& value,
int arrayIndex = -1);`

Sets the property to a string value.

SetProp `inline BOOL SetProp(const char far* name, const char far* value,
int arrayIndex = -1);`

Sets the property to a character string value.

Protected member functions

GetClassName `char far* GetClassName();`

Gets the name of the VBX window class.

GetVBXProperty `BOOL GetVBXProperty(int propIndex, void far* value, int arrayIndex = -1);`

Returns nonzero if the specified property exists. *propIndex* specifies the index value of the integer property whose value you want to get. *value* points to the variable where the value will be stored.

PerformCreate `void PerformCreate(int menuOrId);`

Creates a new control window and associates the VBX control with the window. Establishes the control ID, the VBX control name and class, and the window caption. Sets *Attr.style* to the window style of the control,

Attr.X and *Attr.Y* to the upper-left screen coordinates of the control, and *Attr.W* and *Attr.H* to the width and height of the control.

SetVBXProperty

```
BOOL SetVBXProperty(int propIndex, LONG value, int arrayIndex=-1);
```

Returns nonzero if the specified property value is set, or 0 if unsuccessful. *propindex* is the index number of the property whose value you want to set. *value* is the value to be stored. An optional argument, *arrayIndex*, which is -1 by default, specifies the index value in an array of values of the property to be set.

Response table entries

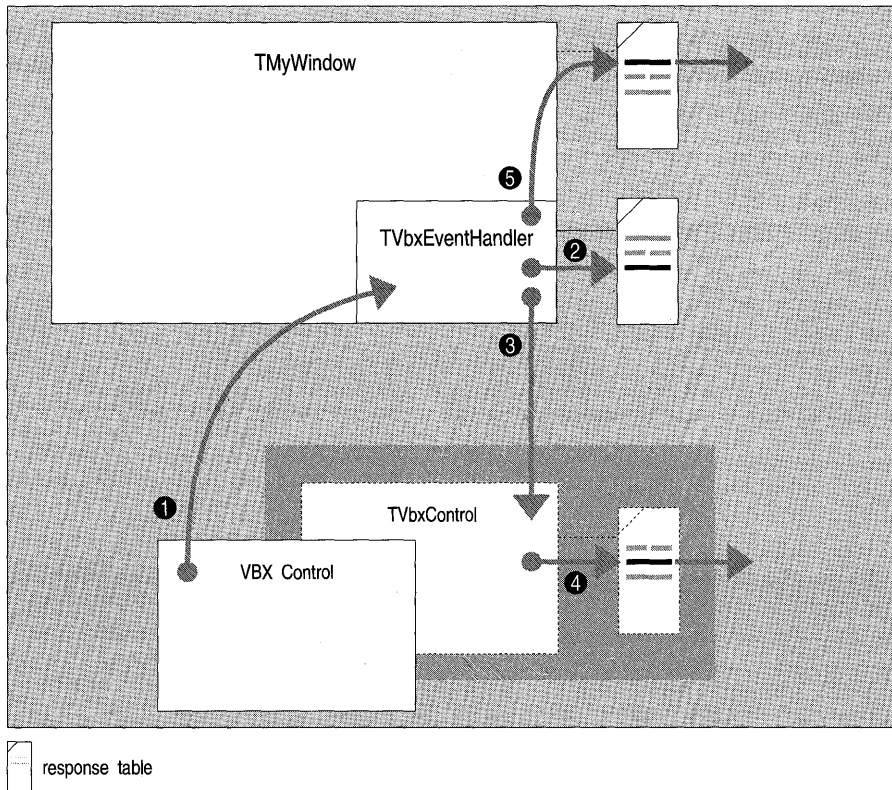
The *TVbxControl* class has no response table entries.

TVbxEventHandler class**vbxctl.h**

Derived from *TEventHandler*, *TVbxEventHandler* handles events from VBX controls. Although you will never need to modify this class, *TVbxEventHandler* needs to be mixed in with your window class so that it can receive events from VBX controls. For example,

```
class TMyWindow:public TWindow, public TVbxEventHandler
{
    // Include class definition here.
}
```

The following diagram illustrates the flow of information between VBX controls, parent windows, and response tables.



When a VBX control fires an event (sends an event message), the following sequence of events occurs:

1. The VBX Control sends a `WM_VBXFIREEVENT` message to *TMyWindow*.
2. *TMyWindow's TVbxEventHandler* finds a `WM_VBXFIREEVENT` message in its response table and calls *EvVbxDispatch*.
3. If a child window is present, *EvVbxDispatch* dispatches the event to the child.
4. If there is an event-handling function in the child window's response table, the child window handles the event.
5. If there is no child window, or the child window doesn't handle the event, *EvVbxDispatch* dispatches the event to *TMyWindow's* response table.

In other words, when a VBX control sends a `WM_VBXFIREEVENT` message, the parent window's *TVbxEventHandler* catches this message first,

converts it into a form understood by a window's response table, and attempts to send the converted message to the child window. If there is no child window or if the child window doesn't handle the message, *TVbxEventHandler* sends the converted message to the parent window. When the parent window receives the message, it calls the handler function that corresponds to the message.

Two response table macros, `EV_VBXEVENTNAME` and `EV_VBXEVENTINDEX`, map VBX events to handler functions. Of the two macros, `EV_VBXEVENTNAME` is more commonly used. `EV_VBXEVENTINDEX` is intended for use with code generators, which can determine the event index values for a VBX control. Both macros call an event handler function and point to the `VBXEVENT` structure. A typical `EV_VBXEVENTNAME` response table entry might be

```
EV_VBXEVENTNAME(IDC_BUTTON1, "MouseMove", EvMouseMove);
```

where `IDC_BUTTON1` is the event ID, "EvMouseMove" is the event name, and *EvMouseMove* is the handler function.

The `lparam` of a `WM_VBX FIREEVENT` message points to a `VBXEVENT` structure, which holds information about the event and the control that generated the event. The `VBXEVENT` structure contains the following members:

```
typedef struct VBXEVENT {
    HCTL    Control;
    HWND    Window;
    int     ID;
    int     EventIndex;
    LPCSTR  EventName;
    int     NumParams;
    LPVOID  ParamList;
} VBXEVENT;
```

where

- *Control* is a handle to the VBX control sending the message.
- *Window* is the handle of the VBX control window.
- *ID* is the ID of the VBX control.
- *EventIndex* is the event index.
- *EventName* is the name of the event.
- *NumParams* holds the number of event arguments.
- *ParamList* is a pointer to a list of pointers to the event's arguments. The *ParamList* data member provides access to the actual arguments of the event.

To handle VBX events, your program uses an event-handling function. In the following example, *EvMouseMove* is the name of the handler function, which passes a pointer to the `VBXEVENT` event structure.

`VBX_EVENTARGNUM` is the macro that takes *event*, *type*, and an index number as its parameters. *event* references the `VBXEVENT` structure, **short** is the event argument type, and 0 and 1 are the index numbers of the argument. The argument types and indexes can be found in the documentation for the VBX control.

```
void EvMouseMove(VBXEVENT FAR* event)
{
    short X = VBX_EVENTARGNUM(event, short, 0);
    short Y = VBX_EVENTARGNUM(event, short, 1);
}
```

Because VBX controls were originally designed to be used with Visual Basic, their event arguments are documented in terms of Basic data types. The following table lists the Basic types, their C++ equivalents, and macros.

Table 1.31
Basic and C++ VBX
data types

Basic	C++	Macro
Boolean	short	<code>VBX_EVENTARGNUM(event, short, index)</code>
Control	HCTL	<code>VBX_EVENTARGNUM(event, HCTL, index)</code>
Double	double	<code>VBX_EVENTARGNUM(event, double, index)</code>
Enum	short	<code>VBX_EVENTARGNUM(event, short, index)</code>
Integer	short	<code>VBX_EVENTARGNUM(event, short, index)</code>
Long	long	<code>VBX_EVENTARGNUM(event, long, index)</code>
Single	float	<code>VBX_EVENTARGNUM(event, float, index)</code>
String	HLSTR	<code>VBX_EVENTARGSTR(event, index)</code>

The following table lists the standard VBX events and corresponding arguments that the Borland C++ VBX emulation library supports.

Table 1.32
VBX event arguments

Event	Arguments
Click	None
DbClick	None
DragDrop	Source as Control, X as Integer, Y as Integer
DragOver	Source as Control, X as Integer, Y as Integer, State as Integer
GotFocus	None
KeyDown	Key as Integer, Shift as Integer
KeyPress	Key as Integer, Shift as Integer
KeyUp	Key as Integer, Shift as Integer
LostFocus	None
MouseDown	X as Integer, Y as Integer
MouseMove	X as Integer, Y as Integer, Shift as Integer, Button as Integer
MouseUp	X as Integer, Y as Integer, Shift as Integer, Button as Integer

For the DragOver event, the state argument can be one of the following values:

- 0, where the source control is being dragged within a target's range.
- 1, where the source control is being dragged out of a target's range.
- 2, where the source control is being moved from one position in the target to another.

For both the DragOver and DragDrop events, the Control argument type should be translated to HCTL (a handle to the VBX control) for C++. The X and Y values are in pixels, not twips.

If a standard VBX event has a Shift key argument, the argument has these bit values:

Table 1.33
Shift key bit values

Key	Bit value
Shift	0x1
Ctrl	0x2
Alt	0x4 (Used in connection with a Menu selection)

If a standard VBX event has a Button key argument, the argument has these bit values:

Table 1.34
Mouse button key arguments

Button	Bit value
Left	0x1
Right	0x2
Middle	0x4

The following example shows how you might use these Shift key arguments. For example, if you want the VBX control to perform some action when the mouse is moved and the Shift key is pressed, you could write a function such as

```
void EvMouseMove(VBXEVENT FAR* event)
{
    short X = VBX_EVENTARGNUM(event, short, 0);
    short Y = VBX_EVENTARGNUM(event, short, 1);
    short Shift = VBX_EVENTARGNUM(event, short, 2);
    short Button = VBX_EVENTARGNUM(event, short, 3);
    if (shift & 0x2)
        MessageBox ("The control key is pressed.");
}
```

Borland C++ uses pixels to express the X and Y coordinate arguments of standard VBX events. This differs from Visual Basic, which expresses coordinates in twips (1/20th of a point or 1/1440 of an inch). Custom

events are usually expressed in terms of twips. You can use these functions to convert between pixels and twips.

Function	Meaning
VBXPix2TwpX()	Converts an X argument from pixels to twips
VBXPix2TwpY()	Converts a Y argument from pixels to twips
VBXTwp2PixX()	Converts an X argument from twips to pixels
VBXTwp2PixY()	Converts a Y argument from twips to pixels

Protected member functions

EvVbxDispatch

```
LRESULT EvVbxDispatch(WPARAM wp, LPARAM lp);
```

After *TVbxEventHandler* receives a `WM_VBXFIREEVENT` message from the parent window, it calls *EvVbxDispatch*, which sends the message to the correct event-handling function and passes a pointer to the `VBXEVENT` structure.

Response table entries

Response table entry	Member function
<code>EV_MESSAGE(WM_VBXFIREEVENT, EvVbxDispatch)</code>	<code>EvVbxDispatch</code>

TView class

docview.h

Derived virtually from both *TEventHandler* and *TStreamableBase*, *TView* is the interface presented to a document so it can access its client views. Views then call the document functions to request input and output streams. Views own the streams and are responsible for attaching and deleting them.

Instead of creating an instance of *TView*, you create a derived class that has access to *TView*'s virtual functions. The derived class must have a way of knowing the associated window (provided by *GetWindow*), of describing the view (provided by *GetViewName*), and of displaying a range of data (*GetSelection*). The view must also be able to restore the view later (*SetSelection*) and to display the document title in its window (*SetDocTitle*).

TView uses several event handler functions to query views, commit, and close views. For example, to query views to find the one with the focus, you

would use the *vnIsWindow* function, passing a handle to the current window. See Chapter 2 for a list of these response functions.

View classes can take various forms. For example, a view class can be a window (through inheritance), can contain a window (an embedded object), can reference a window, or can be contained within a window object. A view class might not even have a window, as in the case of a voice mail or a format converter. Some remote views (for example, those displayed by OLE 2.0 or DDE servers) might not have local windows.

Public data members

Property enum

```
enum {
    PrevProperty = 0,
    ViewClass,
    ViewName,
    NextProperty,
};
```

These property values, defined for *TView*, are available in classes derived from *TView*. *PrevProperty* and *NextProperty* are delimiters for every document's property list. See Chapter 9 in the *ObjectWindows Programmer's Guide* for information about how to use these enumerated property values.

Tag

```
LPOVOID Tag;
```

Tag holds a pointer to the application defined data. Typically, you can use *Tag* to install a pointer to your own application's associated data structure. *TView* zeros *Tag* during construction and doesn't access it again.

Public constructors and destructor

Constructor

```
TView(TDocument& doc);
```

Constructs a *TView* object of the document associated with the view. Sets *ViewId* to *NextViewId*. Calls *TDocument::AttachView* to attach the view to the associated document.

Destructor

```
virtual ~TView();
```

Frees a *TView* object and calls *DetachView* to detach the view from the associated document.

Public member functions

- FindProperty** `virtual int FindProperty(const char far* name);`
FindProperty gets the property index, given the property name (*name*). Returns 0 if the name isn't found.
 See also: *pfxxxx* property access constants
- GetDocument** `TDocument& GetDocument();`
 Returns a reference to the view's document.
- GetNextViewId** `inline static unsigned GetNextViewId();`
 Returns the next view ID to be assigned.
- GetProperty** `virtual int GetProperty(int index, void far* dest, int textlen=0);`
 Returns the total number of properties where *index* is the property index, *dest* contains the property data, and *textlen* is the size of the property array. If *textlen* is 0, property data is returned as binary data; otherwise, property data is returned as text data.
 See also: *pfxxxx* property access constants, *TView::SetProperty*
- GetViewId** `unsigned GetViewId;`
 Returns the unique ID for this view.
- GetViewMenu** `inline TMenuDescr* ViewMenu;`
 Returns the menu descriptor for this view. This can be any existing *TMenuDescr* object. If no descriptor exists, *ViewMenu* is 0.
- GetViewName** `inline virtual LPCSTR GetViewName()=0;`
 Pure virtual function that returns 0. Override this function in your derived class to return the name of the class.
 See also: *TEditView::StaticName*, *TEditView::GetViewName*
- GetWindow** `inline virtual TWindow* GetWindow()`
GetWindow returns the *TWindow* instance associated with the view or 0 if no view exists.
 See also: *TeditView::GetWindow*
- IsOK** `inline BOOL IsOK();`
 Returns nonzero if the view is successfully constructed.
 See also: *TView::NotOK*

TView class

- PropertyCount** inline virtual int PropertyCount();
Gets the total number of properties for the *TDocument* object. Returns *NextProperty* -1.
See also: *pfxxxx property access constants*
- PropertyFlags** virtual int PropertyFlags(int index);
Returns the attributes of a specified property given the index (*index*) of the property whose attributes you want to retrieve.
See also: *pfxxxx property access constants*, *TView::FindProperty*, *TView::PropertyName*
- PropertyName** virtual const char* PropertyName(int index);
Returns the text name of the property given the index value.
See also: *pfxxxx property access constants*, *TView::FindProperty*
- SetDocTitle** inline virtual BOOL SetDocTitle(LPCSTR docname, int index)
Stores the document title.
See also: *TWindow::SetDocTitle*
- SetProperty** inline virtual BOOL SetProperty(int index, const void far* src);
Sets the value of the property, given the index of the property, and *src*, the data type (either binary or text) to which the property must be set.
See also: *pfxxxx property access constants*, *TView::GetProperty*
- SetViewMenu** inline void SetViewMenu(TMenu* menu);
Sets the menu descriptor for this view. This can be any existing *TMenuDescr* object. If no descriptor exists, *ViewMenu* is 0.
See also: *TView::GetViewMenu*

Protected data members

- Doc** TDocument* Doc;
Holds the current document.

Protected member functions

- NotOK** inline void NotOK();

Sets the view to an invalid state, thus causing *IsOK* to return 0.

See also: *TView::IsOK*

TVSlider class

slider.h

Derived from *TSlider*, *TVSlider* provides implementation details for vertical sliders. See *TSlider* for an illustration of a vertical slider.

Public constructors

Constructor

```
TVSlider(TWindow* parent, int id, int X, int Y, int W, int H,
         TResId thumbResId, TModule* module = 0);
```

Constructs a vertical slider object.

Protected member functions

HitTest

```
int HitTest(TPoint& point);
```

Overrides *TSlider's* virtual function and gets information about where a given X, Y location falls on the slider. The return value is in *scrollCodes*.

See also: *TSlider::HitTest*

NotifyParent

```
void NotifyParent(int scrollCode, int pos=0);
```

Overrides *TSlider's* virtual function and sends a WS_VSCROLL message to the parent window.

See also: *TSlider::NotifyParent*

PaintRuler

```
void PaintRuler(TDC& dc);
```

Overrides *TSlider's* virtual function and paints the vertical ruler.

See also: *TSlider::PaintRuler*

PaintSlot

```
void PaintSlot(TDC& dc);
```

Overrides *TSlider's* virtual function and paints the slot in which the thumb slides.

See also: *TSlider::PaintSlot*

PointToPos

```
int PointToPos(TPoint& point);
```

Overrides *TSlider's* virtual function and translates an X,Y point to a position in slider units.

See also: *TSlider::PointToPos*

PosToPoint

```
TPoint PosToPoint(int pos);
```

Overrides *TSlider's* virtual function and translates a position in slider units to an X,Y point.

See also: *TSlider::PosToPoint*

TWidthHeight enum

layoutco.h

```
enum TWidthHeight;
```

Used by the *TLayoutConstraint* struct, *TWidthHeight* enumerates the values that control the width (*lmWidth*) and height (*lmHeight*) of the window.

See also: *TLayoutConstraint* struct

TWindow class

window.h

TWindow, derived from *TEventHandler* and *TStreamableBase*, provides window-specific behavior and encapsulates many of the Windows API functions. *TWindow* functions specify window creation, including registration and attributes.

TWindow is a generic window that can be resized and moved. You can construct an instance of *TWindow*, though normally you'll use *TWindow* as a base for your specialized window classes. In general, to associate and disassociate a *TWindow* object with a window element, you need to follow these steps:

1. Construct an instance of a *TWindow*.
2. Call *Create* or *Execute*, which creates the interface element and then calls *SetupWindow*, which calls the base *SetupWindow* for normal processing, which in turn involves
 - Creating the HWindow and any child HWindows.
 - Calling *TransferData* to setup the transfer of data between the parent and child windows.
3. To destroy the interface element, choose one of the following actions, depending on your application:

- Call *Destroy* to destroy the interface element unconditionally.
 - Call *CloseWindow*, which calls *CanClose* to test whether it's OK to destroy the interface element.
4. There are two ways to destroy the interface object:
- If the object has been **new**'d, call *Delete*.
 - If the object hasn't been **new**'d, the compiler automatically destructs the object.

TWindow is the base class for many classes, including *TFrameWindow*, *TControl*, *TDialog*, and *TMDIChild*. The *ObjectWindows* hierarchy diagram shows the classes that are derived from *TWindow*.

See Chapter 6 in the *ObjectWindows Programmer's Guide* for a description of window objects and Chapter 2, "Event handlers," for a description of the event handlers that respond to Windows messages.

Public data members

Attr	<p><code>TWindowAttr Attr;</code></p> <p><i>Attr</i> holds a <i>TWindowAttr</i> structure, which contains the window's creation attributes. These attributes include the window's style, extended style, position, size, menu ID, child window ID, and menu accelerator table ID.</p> <p>See also: <i>TWindow::TWindow</i>, <i>TWindow::Create</i>, <i>TWindowAttr</i> struct</p>
DefaultProc	<p><code>WNDPROC DefaultProc;</code></p> <p>Holds the address of the default window procedure. <i>DefWindowProc</i> calls <i>DefaultProc</i> to process Windows messages that are not handled by the application.</p> <p>See also: <i>TWindow::DefWindowProc</i></p>
HWindow	<p><code>HWND HWindow;</code></p> <p>Holds the handle to the associated MS-Windows window.</p>
Parent	<p><code>TWindow* parent;</code></p> <p>Points to the interface object that serves as parent window to this interface object.</p>
Scroller	<p><code>TScroller* Scroller;</code></p> <p>Points to the interface object that serves as a scroller for this window.</p>
Status	<p><code>TStatus Status;</code></p>

Status is used to signal an error in the initialization of an interface object. Setting *Status* to a nonzero value causes a *TXIncompatibility* exception to be thrown. Classes derived from *TWindow* do not attempt to associate an interface element with an object whose previous initialization has failed. *Status* is included only to provide backward compatibility with previous versions of *ObjectWindows*.

Title

```
char far* Title;
```

Title points to the window's caption. When there is a valid *HWindow*, *Title* will yield the same information as *::GetWindowText* if you use *TWindow::SetCaption* to set it.

See also: *TDialog::SetCaption*, *TDialog::SetupWindow*, *TWindow::GetWindowTextTitle*, *TWindow::SetCaption*

Public constructors and destructor

Constructor

```
TWindow(HWND hWnd, TModule* module = 0);
```

TWindow has two constructors. This constructor constructs a *TWindow* that is used as an alias for a non-*ObjectWindows* window, and sets *wfAlias*. Unlike the other *TWindow* constructor, this constructor performs the “thunking” and extraction of *HWND* information instead of waiting until the function *Create* creates the interface element.

The following paragraphs describe procedures common to both constructors. *inst* specifies the application or DLL module that owns the *TWindow* instance. *ObjectWindows* needs the correct value of *inst* to find needed resources. If *inst* is 0, *TWindow* sets its module according to the following rules:

- If the window has a parent, the parent's *inst* is copied.
- If the *TWindow* constructor is invoked from an application, the module is set to the application.
- If the *TWindow* constructor is invoked from a DLL that is dynamically linked with the *ObjectWindows* DLL and the currently running application is linked the same way, the library is set to the currently running application.
- If the *TWindow* constructor is invoked from a DLL that is statically linked with the *ObjectWindows* library or the invoking DLL is dynamically linked with *ObjectWindows* DLL but the currently running application is not, no default is used for setting the module. Instead, a *TXInvalidModule* exception is thrown and the object is not created.

See also: *TWindow::Init*, *TWindowFlag* enum

Constructor

```
TWindow(TWindow* parent, const char far* title, TModule* module);
```

Adds *this* to the child list of *parent*, if nonzero, and calls *EnableAutoCreate* so that *this* will be created and displayed along with *parent*. Also sets the title of the window and initializes the window's creation attributes.

See the previous constructor for a description of the procedures common to both constructors.

See also: *TWindow::EnableAutoCreate*

Destructor

```
virtual ~TWindow();
```

Destroys a still-associated interface element by calling *Destroy*. Deletes the window objects in the child list, and then removes *this* from the parent window's child list. Deletes the *Scroller* if it is nonzero. Frees the cursor, if any exists, and the object instance (thunk).

Public member functions

AdjustWindowRect

```
inline static void AdjustWindowRect(TRect& rect, DWORD style, BOOL menu);
```

Calls the Windows API function *::AdjustWindowRect*, which calculates the size of the window rectangle according to the indicated client-rectangle size. *rect* refers to the structure that contains the client rectangle's coordinates. *style* specifies the style of the window. *menu* is TRUE if the window has a menu. See the Windows API online Help for more information about window styles.

See also: *::AdjustWindowRect*

AdjustWindowRectEx

```
inline static void AdjustWindowRectEx(TRect&, DWORD style, BOOL menu,
                                     DWORD exStyle);
```

Calls the Windows API function *::AdjustWindowRectEx*, which calculates the size of a window rectangle that has an extended style. *TRect* refers to the structure that contains the client rectangle's coordinates. *style* specifies the window styles of the window to be adjusted, and *menu* returns TRUE if the window has a menu. *exStyle* indicates the Windows API extended style (for example, *WS_EX_TOPMOST*) to be used for the window. See the Windows API online Help for additional information about extended window styles.

See also: *::AdjustWindowRectEx*, *TWindowsAttr* struct

BringWindowToTop

```
inline void BringWindowToTop();
```


Calls the Windows API function `::BringWindowToTop`, which brings a pop-up or child window to the top of the stack of overlapping windows and activates it.

See also: `::BringWindowToTop`

CanClose

```
virtual BOOL CanClose();
```

Returns TRUE if the associated interface element can be closed. Calls the `CanClose` member function of each of its child windows. Returns FALSE if any of the `CanClose` calls returns FALSE.

CheckDlgButton

```
inline void CheckDlgButton(int buttonId, UINT check);
```

Calls the Windows API function `::CheckDlgButton`, which places a checkmark in (or removes a checkmark from) the button specified in `buttonId`. If `check` is nonzero, the checkmark is placed next to the button; if 0, the checkmark is removed. For buttons having three states, `check` can be 0 (clear), 1 (checked), or 2 (gray).

See also: `::CheckDlgbutton`

CheckRadioButton

```
inline void CheckRadioButton(int firstButtonId, int lastButtonId,
                             int checkButtonId);
```

Calls the Windows API function `::CheckRadioButton`, which checks the radio button specified by `checkButtonId` and removes the checkmark from the other radio buttons in the group. `firstButtonId` and `lastButtonId` specify the first and last buttons, respectively, in the group.

See also: `::CheckRadioButton`

ChildBroadcastMessage `ChildBroadcastMessage(UINT msg, WPARAM wParam, LPARAM lParam);`

Sends a message to all immediate children using `SendMessage`.

See also: `TWindow::SendMessage`

ChildWindowFromPoint `inline HWND ChildWindowFromPoint(const TPoint&) const;`

Calls the Windows API function `::ChildWindowFromPoint`, which determines which of the child windows contains the point specified in `TPoint`. Returns a handle to the window that contains the point, or 0 if the point lies outside the parent window.

See also: `::ChildWindowFromPoint`, `TWindow::WindowFromPoint`

ChildWithid

```
TWindow* ChildWithid(int id);
```

Returns a pointer to the window in the child window list that has the supplied ID. Returns 0 if no child window has the indicated `id`.

CleanUpWindow

```
virtual void CleanupWindow();
```

CleanupWindow gives derived classes a chance to clean up hwnd-related resources. Derived classes should call the base class's version of *CleanupWindow* as the last step before returning to the parent class.

ClearFlag

```
inline void ClearFlag(TWindowFlag mask);
```

Clears the *TWindow* *wfXxxx* constant flags (for example *wfAlias*, *wfTransfer*, and so on).

See also: *TWindowFlag* enum

ClientToScreen

```
void ClientToScreen(TPoint&) const;
```

Converts the client coordinates specified in *TPoint* to screen coordinates for the new window.

CloseWindow

```
void CloseWindow();
```

After determining that it's OK to close the window, *CloseWindow* calls *Destroy* to destroy the hwnd. If *this* is the main window of the application, calls *GetApplication()->CanClose*. Otherwise, calls *this->CanClose* to determine whether the window can be closed.

See also: *TApplication::CanClose*, *TWindow::CanClose*

CmExit

```
void CmExit();
```

CmExit is called in response to the selection of a menu item that has an ID of *CM_EXIT*. If *this* is the main window, *CmExit* calls *CloseWindow*.

Create

```
virtual BOOL Create();
```

Creates the windows interface element to be associated with this *ObjectWindows*'s interface element.

CreateCaret

```
inline void CreateCaret(HBITMAP);
```

Calls the Windows API function *::CreateCaret*, which creates a new caret for the system. *HBITMAP* specifies the bitmapped caret shape.

See also: *::CreateCaret*

CreateCaret

```
inline void CreateCaret(BOOL isGray, int width, int height);
```

Calls the Windows API function *::CreateCaret* to create a new caret for the system with the specified shape, bitmap shade, width, and height. If *width* or *height* is 0, the corresponding system-defined border size is used.

See also: *::CreateCaret*

CreateChildren

```
BOOL CreateChildren();
```

Creates the child windows in the child list whose auto-create flags (with *wfAutoCreate* mask) are set.

See also: *TWindow::EnableAutoCreate*, *TWindow::DisableAutoCreate*, *TWindowFlag* enum

DefaultProcessing

```
LRESULT DefaultProcessing();
```

DefaultProcessing serves as a general-purpose default processing function that handles a variety of messages. Before calling *DefaultProcessing*, however, a window completes this sequence of events:

- If the window is already created, *SubclassWindow* is used to install *StdWndProc* in place of the window's current procedure.
- If the window hasn't been created, *InitWndProc* is set up as the window proc in the class. Then, when the window first receives a message, *InitWndProc* calls *GetModule* to get the window's instance thunk (created by the constructor by calling *CreateInstanceThunk*). *InitWndProc* then switches the message-receiving capability from the window's procedure to *StdWndProc*.

After this point, *StdWndProc* responds to incoming messages by calling the window's virtual *WindowProc* to process the messages. *ObjectWindows* uses the special registered message *GetWindowPtrMsgId* to get the **this** pointer of an HWND. *StdWndProc* responds by returning the **this** pointer obtained from the thunk.

If the incoming message is not a command, *WindowProc* immediately searches the window's response table for a matching entry. If the incoming message is a command, *WindowProc* calls *EvCommandEnable* or *EvCommand*. *EvCommandEnable* and *EvCommand* begin searching for a matching entry in the focus window's response table. If an entry is found, the corresponding function is dispatched; otherwise *ObjectWindows* calls *DefaultProcessing* to finish the recursive walk back up the parent chain, searching for a match until the receiving window (the window that initially received the message) is reached. At this point, one of the following actions occurs:

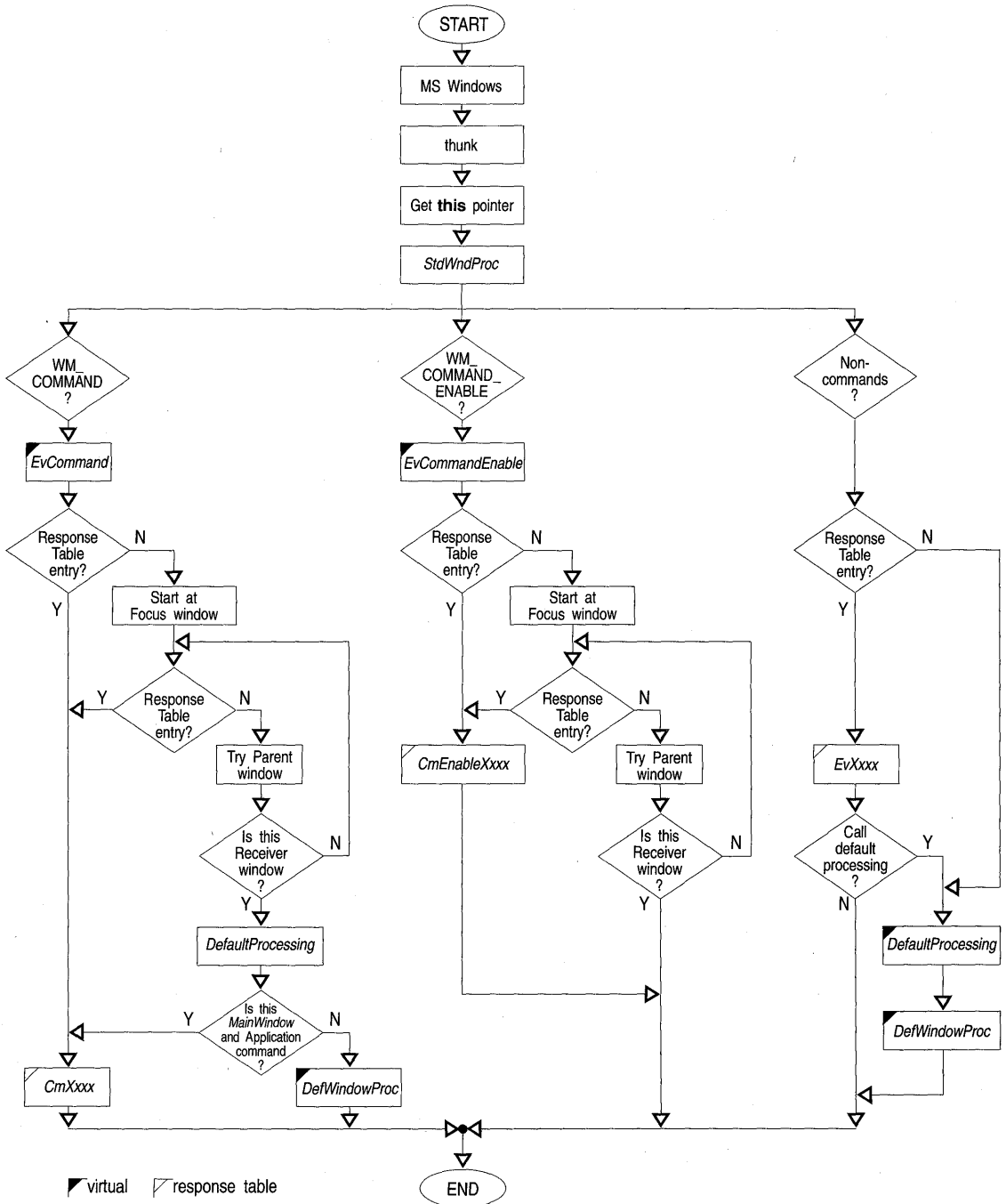
- If there is still no match and this is the *MainWindow* of the application, the window searches the application's response table.
- If there are no matches and this is a command, *DefWindowProc* is called.
- If this is a *CommandEnable* message, no further action is taken.
- If this is not a command, and if a response table entry exists for the window, *WindowProc* dispatches the corresponding *EvXxxx* function to handle the message.
- If this is the application's *MainWindow*, and the message is designed for the application, the message is forwarded to the application.
- For any other cases, the window calls *DefWindowProc*.

To perform a DLL test for a non-ObjectWindows application, you might process the following messages.

```
LRESULT FAR PASCAL _export
WndProc (HWND hWnd, UINT message, WPARAM, wParam, LPARAM lParam)
{
    switch (message) {
        case WM_DESTROY:
            PostQuitMessage (0);
            break;
        case WM_COMMAND:
            //Insert processing statements here.//
    } else
        return DefWindowProc (hWnd, message, wParam, lParam);
    break;
default:
    return DefWindowProc (hWnd, message, wParam, lParam);
}
```

See also: *TWindow::DefWindowProc*

The following diagram illustrates this sequence of message-processing events:



- DefWindowProc** virtual LRESULT DefWindowProc(UINT msg, WPARAM wParam, LPARAM lParam);
- Calls *DefaultProc*, the subclassed window procedure, for default Windows processing and passes the incoming Windows message. You usually don't need to call this function directly. Classes such as *TMDIFrame* and *TMDIChild's DefWindowProc* override this function to perform specialized default processing.
- See also: *TWindow::DefaultProc*, *TWindow::WindowProc*, *TMDIFrame::DefWindowProc*
- Destroy** virtual void Destroy(int retVal = 0);
- First, *Destroy* calls *EnableAutoCreate* for each window in the child list to ensure that windows in the child list will be re-created if *this* is re-created. Then, it destroys the associated interface element.
- If a derived window class expects to be destructed directly, it should call *Destroy* as the first step in its destruction so that any virtual functions and event handlers can be called during the destroy sequence.
- See also: *TWindow::EvDestroy*, *TWindow::EnableAutoCreate*
- DestroyCaret** inline static void DestroyCaret();
- Calls the Windows API function *DestroyCaret*, which first checks the ownership of the caret. If a window in the current task owns the caret, *DestroyCaret* destroys the caret and removes it from the screen.
- See also: *TWindow::CreateCaret*, *::DestroyCaret*
- DisableAutoCreate** inline void DisableAutoCreate();
- Disables the feature that allows an associated child window interface element to be created and displayed along with its parent window. Call *DisableAutoCreate* for pop-up windows and controls if you want to create and display them at a time later than their parent windows.
- See also: *TWindow::EnableAutoCreate*
- DisableTransfer** inline void DisableTransfer();
- Disables (for the interface object) the transfer mechanism, which allows state data to be transferred to and from a transfer buffer.
- See also: *TWindowFlag enum*
- DragAcceptFiles** void DragAcceptFiles(BOOL accept);
- If a window can process dropped files, *DragAcceptFiles* calls the Windows API function *DragAcceptFiles* to set *accept*.

See also: *::DragAcceptFiles*

DrawMenuBar

```
inline void DrawMenuBar();
```

Calls the Windows API function *DrawMenuBar* to redraw the menu bar. This function should be called to redraw the menu if the menu is changed after the window is created.

See also: *::DrawMenuBar*

EnableAutoCreate

```
inline void EnableAutoCreate();
```

Ensures that an associated child-window interface element is created and displayed along with its parent window. By default, this feature is enabled for windows and controls, but disabled for dialog boxes.

See also: *TWindow::DisableAutoCreate*

EnableTransfer

```
inline void EnableTransfer();
```

Enables the transfer mechanism, which allows state data to be transferred between the window and a transfer buffer.

See also: *TWindow::DisableTransfer*

EnumProps

```
inline int EnumProps(PROPENUMPROC proc);
```

Calls the Windows API function *::EnumProps* to enumerate the items in the property list of the specified window.

See also: *::EnumProps*

EvChildInvalid

```
void EvChildInvalid(HWND);
```

Responds to a WM_CHILDINVALID message posted by a child edit control. Indicates that the contents of the child window are invalid.

EvCommand

```
virtual LRESULT EvCommand(UINT id, HWND hWndCtl, UINT notifyCode);
```

WindowProc calls *EvCommand* to handle WM_COMMAND messages. *id* is the identifier of the menu item or control. *hWndCtl* holds a value that represents the control sending the message. If the message is not from a control, it is 0. *notifyCode* holds a value that represents the control's notification message. If the message is from an accelerator, *notifyCode* is 1; if the message is from a menu, *notifyCode* is 0.

See also: *TWindow::DefaultProcessing*

EvCommandEnable

```
virtual void EvCommand(TCommandEnabler &);
```

Called by *WindowProc* to handle WM_COMMAND_ENABLE messages, *EvCommand* calls the *CmXxxx* command-handling function or calls *DefaultProcessing* to handle the incoming message.

See also: *TWindow::DefaultProcessing*

EvSysCommand

```
void EvSysCommand (UINT cmdType, TPoint &);
```

Responds to a user-selected command from the System menu or when the user selects the maximize or minimize box. Applications that modify the system menu must process *EvSysCommand* messages. Any *EvSysCommand* messages not handled by the application must be passed to *DefWindowProc*. The parameter *cmdType* can be one of the following system commands:

Constant	Meaning
SC_CLOSE	Close the window.
SC_HOTKEY	Activate the specified window.
SC_HSCROLL	Scroll horizontally.
SC_KEYMENU	Retrieve a menu through a keystroke.
SC_MAXIMIZE (or SC_ZOOM)	Maximize the window.
SC_MINIMIZE (or SC_ICON)	Minimize the window.
SC_MOUSEMENU	Retrieve a menu through a mouse click.
SC_NEXTWINDOW	Move to the next window.
SC_PREVWINDOW	Move to the previous window.
SC_SCREENSAVE	Execute the specified screen saver.
SC_SIZE	Size the window
SC_TASKLIST	Activate the Windows Task Manager.
SC_VSCROLL	Scroll vertically.

In the following example, *EvSysCommand* either processes system messages or calls *DefaultProcessing*:

```
void MyWindow::EvSysCommand(UINT cmdType, TPoint&)
{
    switch (cmdType & 0xFFFF0) {
        case SC_MOUSEMENU:
        case SC_KEYMENU:
            break;
        default:
            DefaultProcessing();
    }
}
```

See also: *TWindow::DefaultProcessing*

FirstThat

```
TWindow* FirstThat(TCondFunc test, void* paramList = 0);
```

There are two *FirstThat* functions, both of which pass a pointer to an iterator function. This *FirstThat* points to a nonmember function as its first parameter; the other *FirstThat* (see the following entry) points to a member function instead.

Both *FirstThat* functions iterate over the child list, calling a Boolean *test* function and passing each child window in turn as an argument (along with *paramList*). If a *test* call returns TRUE, the iteration is stopped and *FirstThat* returns the child window object that was supplied to *test*. Otherwise, *FirstThat* returns 0.

In the following example, *GetFirstChecked* calls *FirstThat* to obtain a pointer (*p*) to the first check box in the child list that is checked:

```

    BOOL IsThisBoxChecked(TWindow* p, void*) {
        return ((TCheckBox*)p)->GetCheck() == BF_CHECKED;
    }

    TCheckBox* TMyWindow::GetFirstChecked() {
        return FirstThat(IsThisBoxChecked);
    }

```

See also: *TCondFunc* type

FirstThat

```
TWindow* FirstThat(TCondMemFunc test, void* paramList = 0);
```

Refer to the previous description of *FirstThat*. The difference between the two *FirstThat* functions is that the previous *FirstThat* takes a nonmember function as a parameter and this *FirstThat* takes a member function as a parameter.

See also: *TCondMemFunc* type

FlashWindow

```
inline BOOL FlashWindow(BOOL invert);
```

Calls the Windows API function *FlashWindow* to change the window from active to inactive or vice versa. If *invert* is nonzero, the window is flashed. If *invert* is 0, the window is returned to its original state—either active or inactive.

See also: *::FlashWindow*

ForEach

```
void ForEach(TActionFunc action, void* paramList = 0);
```

There are two *ForEach* functions. This *ForEach* takes a nonmember function as its first parameter; the other *ForEach* (see the following entry) takes a member function instead. Both versions of *ForEach* iterate over the child list, calling a nonmember function supplied as the *action* to be performed and passing each child window in turn as the argument (along with *paramList*).

In the following example, *CheckAllBoxes* calls *ForEach*, checking all the check boxes in the child list:

```

    void CheckTheBox(TWindow* p, void*) {
        ((TCheckBox*)p)->Check();
    }

```

```

    }
    void CheckAllBoxes() {
        ForEach(CheckTheBox);
    }

```

See also: *TActionFunc* type

ForEach

```
void ForEach(TActionMemFunc action, void* paramList = 0);
```

Refer to the previous *ForEach* description. The difference between the two *ForEach* members is that the first *ForEach* takes a nonmember function as a parameter and this *ForEach* takes a member function as a parameter.

See also: *TActionMemFunc* type

ForwardMessage

```
inline LRESULT ForwardMessage(BOOL send = TRUE)
```

Forwards the window's current message. Calls the Windows API function *::SendMessage* if *send* is *TRUE*; otherwise calls *::PostMessage*.

See also: *TWindow::PostMessage*

ForwardMessage

```
LRESULT ForwardMessage(HWND hWnd, BOOL send = TRUE);
```

Forwards the window's current message to another *HWND*. Calls the Windows API function *::SendMessage* if *send* is *TRUE*; otherwise calls *::PostMessage*.

See also: *TWindow::PostMessage*

GetActiveWindow

```
inline static HWND GetActiveWindow();
```

Calls the Windows API function *GetActiveWindow* to retrieve the handle of the active window. Returns 0 if no window is associated with the calling thread.

See also: *::GetActiveWindow*

GetApplication

```
TApplication* GetApplication();
```

Gets a pointer to the *TApplication* object associated with **this**.

GetCaretBlinkTime

```
inline static UINT GetCaretBlinkTime();
```

Calls the Windows API function *GetCaretBlinkTime* to retrieve the caret blink rate in milliseconds.

See also: *TWindow::SetCaretBlinkTime*

GetCaretPos

```
inline static void GetCaretPos(TPoint& point);
```

Calls the Windows API function *GetCaretPos* to get the position of the caret in the coordinates of the client window. *point* refers to the structure that receives the client coordinates of the caret.

See also: *TWindow::SetCaretPos*

GetChildren

```
void GetChildren(ipstream &is);
```

Reads child windows from the supplied stream into a child list. *GetChildren* should be called only during a *read* operation to read child windows that were written to the stream by *PutChildren*.



This member function assumes that the child list is currently empty.

GetClassLong

```
inline long GetClassLong(int index) const;
```

Calls the Windows API function *GetClassLong* to retrieve the long value from the Windows API WNDCLASS structure. Depending on the value of *index*, *GetClassLong* retrieves a handle to the background brush, cursor, icon, module, menu, window function, and so on.

See also: *::GetClassLong*, *Twindow::SetClassLong*

GetClassWord

```
inline WORD GetClassWord(int index) const;
```

Calls the Windows API function *GetClassWord* to get the word value from the Windows API WNDCLASS structure. Depending on the value of *index*, *GetClassLong* can retrieve class, window, or style information.

See also: *::GetClassWord*, *TWindow::SetClassWord*

GetClientRect

```
inline TRect GetClientRect() const;
```

Calls the Windows API function *GetClientRect* to get the coordinates of the window's client area.

See also: *::GetClientRect*

GetClientRect

```
inline void GetClientRect(TRect &) const;
```

Calls the Windows API function *GetClientRect* to get the coordinates of the window's client area and to copy them into the object referred to by *TRect*.

See also: *::GetClientRect*

GetCursorPos

```
inline static void GetCursorPos(TPoint &pos);
```

Calls the Windows API function *GetCursorPos* to retrieve the cursor's current position (in client window screen coordinates) and copy them into the structure pointed to by *TPoint*.

See also: *::GetCursorPos*

GetDesktopWindow

```
inline static HWND GetDesktopWindow;
```

Calls the Windows API function *GetDesktopWindow* and returns a handle to the desktop window.

- GetDlgItem** `HWND GetDlgItem(int childId) const;`
 Calls the Windows API function *GetDlgItem* to retrieve the handle of a control specified by *childId*.
 See also: *TWindow::GetDlgItem*
- GetDlgItemInt** `UINT GetDlgItemInt(int childId, BOOL *translated = 0, BOOL isSigned = TRUE) const;`
 Calls the Windows API function *GetDlgItemInt* to retrieve the text of a control specified by *childId*. *translated* points to the variable that receives the translated value. *isSigned* indicates that the retrieved value is signed (the default).
 See also: *TWindow::GetDlgItemInt*
- GetDlgItemText** `UINT GetDlgItemText(int childId, char far* text, int max) const;`
 Calls the Windows API function *GetDlgItemText* to retrieve the text of a control specified by *childId*. *text* points to the text buffer to receive the text. *max* specifies the maximum length of the caption, which is truncated if it exceeds this length.
 See also: *TWindow::SetDlgItemText*, *::GetDlgItemText*
- GetFirstChild** `inline TWindow* GetFirstChild()`
 Returns a pointer to the first child window in the interface object's child list.
 See also: *TWindowAttr* struct
- GetId** `int GetId();`
 Returns *Attr.Id*, the ID used to find the window in a specified parent's child list.
 See also: *TWindowAttr* struct
- GetModule** `inline TModule* GetModule() const;`
 Returns a pointer to the module object.
- GetLastActivePopup** `HWND GetLastActivePopup() const;`
 Returns the last active popup window in the list.
- GetLastChild** `TWindow* GetLastChild();`
 Returns a pointer to the last child window in the interface object's child list.
- GetMenu** `inline HMENU GetMenu() const;`

Calls the Windows API function *GetMenu* to get the handle to the menu of the indicated window. If the window has no menu, the return value is 0.

See also: *TWindow::SetMenu*

GetNextDlgGroupItem inline HWND GetNextDlgGroupItem(HWND hWndCtrl, BOOL previous = FALSE) const;

Calls the Windows API function *GetNextDlgGroupItem* to return either the next or the previous control in the dialog box. *hWndCtrl* identifies the control in the dialog box where the search begins. If *previous* is 0, *GetNextDlgGroupItem* searches for the next control; if nonzero, it searches for the previous control.

See also: *::GetNextDlgGroupItem*

GetNextDlgTabItem inline HWND GetNextDlgTabItem(HWND hWndCtrl, BOOL previous = FALSE) const;

Calls the Windows API function *GetNextDlgTabItem* to return the handle of the first control that lets the user press the TAB key to move to the next control (that is, the first control with the WS_TABSTOP style associated with it). *hWndCtrl* identifies the control in the dialog box where the search begins. If *previous* is 0, *GetNextDlgGroupItem* searches for the next control; if nonzero, it searches for the previous control.

See also: *::GetNextDlgTabItem*

GetNextWindow inline HWND GetNextWindow(UINT dirFlag) const;

Calls the Windows API function *GetNextWindow* to find the handle associated with either the next or previous window in the window manager's list. *dirFlag* specifies the direction of the search. Under the WIN32 API, *GetNextWindow* returns either the next or the previous window's handle. If the application is not running under Win32, *GetNextWindow* returns the next window's handle.

See also: *::GetNextWindow*

GetWindowPtr void GetWindowPtr(HWND hWnd);

Given the handle to a window (*hWnd*), *GetWindowPtr* returns the *TWindow* pointer associated with the window.

GetParent inline HWND GetParent() const;

Calls the Windows API function *GetParent* to retrieve the handle of the parent window. If none exists, returns 0.

See also: *TWindow::SetParent*

GetProp inline HANDLE GetProp(WORD atom) const;

Calls the Windows API function *GetProp* to return a handle to the property list of the specified window. *atom* contains a value that identifies the character string whose handle is to be retrieved.

See also: *TWindow::SetProp, ::GetProp*

GetProp

```
inline HANDLE GetProp(const char far* string) const;
```

Calls the Windows API function *GetProp* to return a handle to the property list of the specified window. Unlike the previous *GetProp* function, *string* points to the string whose handle is to be retrieved.

See also: *TWindow::SetProp, ::GetProp*

GetScrollPos

```
inline int GetScrollPos(int bar) const;
```

Calls the Windows API function *GetScrollPos* to return the thumb position in the scroll bar. *bar* identifies the position (horizontal, vertical, or scroll bar control) to return; it can be one of the Windows API *SB_xxxx* constants.

See also: *TWindow::GetScrollPos, ::GetScrollPos*

GetScrollRange

```
inline void GetScrollRange(int bar, int &minPos, int &maxPos) const;
```

Calls the Windows API function *GetScrollRange* to return the thumb position in the scroll bar. *bar* identifies the position (horizontal, vertical, or scroll bar control) to return; it can be one of the Windows API *SB_xxxx* constants. *minPos* and *maxPos* specify the lower and upper range, respectively, of the scroll bar positions.

See also: *TWindow::SetScrollRange, ::GetScrollRange*

GetSysModalWindow `inline static HWND GetSysModalWindow();`



Retrieves the handle of the system-modal window.

See also: *TWindow::SetSysModalWindow*

GetSystemMenu

```
inline HMENU GetSystemMenu(BOOL revert = FALSE) const;
```

Calls the Windows API function *GetSystemMenu* and returns a handle to the system menu so that an application can access the system menu.

See also: *::GetSystemMenu*

GetTopWindow

```
inline HWND GetTopWindow()const;
```

Calls the Windows API function *GetTopWindow*, which returns a handle to the top window currently owned by this parent window. If no children exist, *GetTopWindow* returns 0.

See also: *::GetTopWindow*

GetUpdateRect

```
inline BOOL GetUpdateRect(TRect &, BOOL erase = TRUE) const;
```

Calls the Windows API function *GetUpdateRect* to retrieve the screen coordinates of the rectangle that encloses the updated region of the specified window. *erase* specifies whether *GetUpdateRect* should erase the background of the updated region.

See also: *::GetUpdateRect*, *TWindow::RedrawWindow*

GetUpdateRgn

```
inline BOOL GetUpdateRgn(TRegion &, BOOL erase = TRUE) const;
```

Calls the Windows API function *GetUpdateRgn* to copy a window's update region into a region specified by *region*. *erase* specifies whether *GetUpdateRgn* should erase the background of the updated region. *GetUpdateRgn* returns a value indicating the type of region such as *COMPLEXREGION*, *ERROR*, *NULLREGION*, or *SIMPLEREGION*. See the Windows API online Help for a description of these values.

See also: *TWindow::RedrawWindow*, *::GetUpdateRgn*

GetWindowLong

```
inline long GetWindowLong(int index) const;
```

Returns information (such as style attributes) about the indicated window from the *WNDCLASS* structure.

See also: *TWindow::GetClassLong*, *::GetWindowLong*

GetWindowPlacement `inline BOOL GetWindowPlacement(WINDOWPLACEMENT* place) const;`

Calls the Windows API function *GetWindowPlacement* to retrieve display and placement information (normal, minimized, and maximized positions) about the window and stores that information in *place*.

See also: *TWindow::SetWindowPlacement*, *TWindow::Show*

GetWindowRect

```
inline void GetWindowRect(TRect &) const;
```

Gets the screen coordinates of the window's rectangle and copies them into the area pointed to by *TRect*.

See also: *TWindow::GetClientRect*, *::GetWindowRect*

GetWindowTask

```
inline HANDLE GetWindowTask() const;
```



Calls the Windows API function *GetWindowThreadProcessId* to return the thread ID of the thread that created the specified window.

GetWindowTask

```
inline HTASK GetWindowTask() const;
```



Calls the Windows API function *GetWindowTask* to return a handle to the task that created the specified window.

GetWindowText

```
inline int GetWindowText(char far* string, int maxCount) const;
```

Calls the Windows API function *GetWindowText* which copies the window's title into a buffer pointed to by *string*. *maxCount* indicates the number of characters to copy into the buffer. A string of characters longer than *maxCount* is truncated. *GetWindowText* returns the length of the string or 0 if no title exists.

See also: *TWindow::SetWindowText*, *TWindow::GetWindowTextTitle*, *::GetWindowText*, *TWindow::SetCaption*

GetWindowTextLength inline int GetWindowTextLength() const;

Calls the Windows API function *GetWindowTextLength*, which returns the length of the specified window's title.

See also: *TWindow::SetWindowText*, *::GetWindowTextLength*

GetWindowTextTitle void GetWindowTextTitle();

Updates the *TWindow* title data member (*Title*) from the current window's caption. *GetWindowTextTitle* is used to keep *Title* synchronized with the actual window state when there is a possibility that the state might have changed.

See also: *TWindow::SetCaption*, *TWindow::Title*

GetWindowWord inline WORD GetWindowWord(int index) const;

Calls the Windows API function *GetWindowWord*, which retrieves information about the window depending on the value stored in *index*. *GetWindowWord* returns values that indicate the handle of the module that owns the window (GWW_HMODULE), the handle of the parent window (GWW_HWNDPARENT), or the parent window (GWW_HWNDPARENT).

See also: *TWindow::GetWindowLong*, *TWindow::SetWindowWord*, *TWindow::SetParent*, *::GetWindowWord*

HandleMessage LRESULT HandleMessage(UINT msg, WPARAM wParam = 0, LPARAM lParam = 0);

Handles message sent to a window. *HandleMessage* can be called directly to handle Windows messages without going through *SendMessage*.

See also: *TWindow::SendMessage*

HideCaret inline void HideCaret();

Calls the Windows API function *HideCaret*, which removes the caret from the specified display screen. The caret is hidden only if the current task's window owns the caret.

See also: *TWindow::CreateCaret*, *::HideCaret*, *TWindow::ShowCaret*

HiliteMenuItem

```
inline BOOL HiliteMenuItem(HMENU, UINT idItem, UINT hilite);
```

Calls the Windows API function *HiliteMenuItem*, which either highlights or removes highlighting from a top-level item in the menu. *idItem* indicates the menu item to be processed. *hilite* (which contains a value that indicates if the *idItem* is to be highlighted or is to have the highlight removed) can be one or more of the *MF_xxxx* constants (for example, *MF_HILITE* or *MF_BYPOSITION*).

See also: *TWindow::HiliteMenuItem*

HoldFocusHwnd

```
virtual BOOL HoldFocusHwnd(HWND hWndLose, HWND hWndGain);
```

Responds to a request by a child window to hold its HWND when it is losing focus. Stores the child's HWND in *HwndRestoreFocus*.

See also: *TFrameWindow::HoldFocusHwnd*

Invalidate

```
inline void Invalidate(BOOL erase = TRUE);;
```

Calls the Windows API function *InvalidateRect* to invalidate (mark for painting) the entire client area. By default, the background of the client area is marked for erasing.

See also: *TWindow::Validate, ::InvalidateRect*

InvalidateRect

```
inline void InvalidateRect(const TRect&, BOOL erase = TRUE);;
```

Calls the Windows API function *InvalidateRect* to invalidate a specified client area. By default, the background of the client area to be invalidated is marked for erasing.

See also: *TWindow::ValidateRect, ::InvalidateRect*

InvalidateRgn

```
inline void InvalidateRgn(HRGN hRgn, BOOL erase = TRUE);
```

Calls the Windows API function *InvalidateRgn* to invalidate a client area within a region specified by *hRgn* when the application receives a *WM_PAINT* message. By default, the background within the region is marked for erasing.

See also: *TWindow::ValidateRgn, ::InvalidateRgn*

IsChild

```
inline BOOL IsChild(HWND) const;
```

Is TRUE if the window is a child of this window.

IsDlgButtonChecked

```
inline UINT IsDlgButtonChecked(int buttonId) const;
```

Is TRUE if the child button specified in *buttonId* is checked.

IsFlagSet

```
inline BOOL IsFlagSet(TWindowFlag mask);
```

Returns the state of the bit flag in *Attr.Flags* whose *mask* is supplied. Returns TRUE if the bit flag is set, and FALSE if not set.

See also: *TWindowAttr* struct

IsIconic

```
inline BOOL IsIconic() const;
```

Is TRUE if window is iconic or minimized.

IsWindowVisible

```
inline BOOL IsWindowVisible() const;
```

Is TRUE if the window is visible. By default, *TWindow's* constructor sets the window style attribute (WS_VISIBLE) so that the window is visible.

IsZoomed

```
inline BOOL IsZoomed() const;
```

Is TRUE if window is zoomed or maximized.

KillTimer

```
inline BOOL KillTimer(UINT timerId);
```

Calls the Windows API function *KillTimer* which gets rid of the timer and removes any WM_TIMER messages from the message queue. *timerId* contains the ID number of the timer event to be killed.

See also: *TWindow::SetTimer*, *::KillTimer*

LockWindowUpdate

```
inline BOOL LockWindowUpdate();
```

Calls the Windows API function *LockWindowUpdate* to prevent any drawing in the specified window. Returns 0 if the window is locked.

See also: *::LockWindowUpdate*

MapWindowPoints

```
inline void MapWindowPoints(HWND hWndTo, TPoint* points, int count) const;
```

Calls the Windows API function *MapWindowPoints* to map a set of points in one window to a relative set of points in another window. *hWndTo* specifies the window to which the points are converted. *points* points to the array containing the points. If *hWndTo* is 0, the points are converted to screen coordinates. *count* specifies the number of *points* structures in the array.

See also: *::MapWindowPoints*

MessageBox

```
int MessageBox(const char far* text, const char far* caption = 0,
               UINT type = MB_OK);
```

Creates and displays a message box that contains a message (*text*), a title (*caption*), and icons or push buttons (*type*). If *caption* is 0, the default title is displayed. Although *type* is set to one push button by default, it can contain a combination of the WIN API MB_XXX constants. This function returns one of the following WIN API constants: IDABORT, IDCANCEL, IDIGNORE, IDNO, IDOK, IDRETRY, IDYES.

See also: *::MessageBox*

MoveWindow

```
inline void MoveWindow(int x, int y, int w, int h, BOOL repaint = FALSE);
```

Calls the Windows API function *MoveWindow* to reposition the specified window. *x* and *y* specify the new upper left coordinates of the window; *w* and *h* specify the new width and height, respectively. If *repaint* is *FALSE*, the window is not repainted after it is moved.

See also: *::MoveWindow*

MoveWindow

```
inline void MoveWindow(const TRect &, BOOL repaint = FALSE);
```

Calls the Windows API function *MoveWindow*, passing the left and top coordinates and the width and height of the new screen rectangle.

See also: *::MoveWindow*

Next

```
inline TWindow* Next();
```

Returns a pointer to the next window in the window's sibling list.

See also: *TWindow::Previous*

NumChildren

```
unsigned NumChildren();
```

Returns the number of *ObjectWindows* child windows.

OpenClipboard

```
inline TClipboard &OpenClipboard();
```

Calls the *TClipboard* function *GetClipboard* to get the global Clipboard object. Then, *OpenClipboard* opens the Clipboard object before returning it.

See also: *TClipboard::GetClipboard*

Paint

```
virtual void Paint(TDC&, BOOL erase, TRect&);
```

Called by base classes when responding to a *WM_PAINT* message, *Paint* serves as a placeholder for derived types that define *Paint* member functions. *Paint* is called by *EvPaint* and requested automatically by Windows to redisplay the window's contents. *TDC* is the paint display context supplied to text and graphics output functions. The supplied reference to the *TRect* structure is the bounding rectangle of the area that requires painting. *erase* indicates whether the background needs erasing.

PostMessage

```
inline BOOL PostMessage(UINT msg, WPARAM wParam = 0, LPARAM lParam=0)
    const;
```

Wrapper function for the Window's API function by the same name, *PostMessage* posts a message (*msg*) in a window's message queue. It returns without waiting for the corresponding window to process the message.

See also: *TWindow::ForwardMessage*, *::PostMessage*

- PerformCreate** `virtual void PerformCreate(int menuOrId);`
- Create an MS_Windows interface element to be associated with an ObjectWindows window.
- PreProcessMsg** `virtual BOOL PreProcessMsg(MSG& msg);`
- PreProcessMsg* allows preprocessing of window messages and keyboard messages. If you override this method in a derived class, be sure to call *PreProcessMsg* because it handles the translation of accelerator keys. When nonzero is returned, message processing stops.
- See also: *TApplication::ProcessAppMsg*
- Previous** `TWindow* Previous();`
- Returns a pointer to the previous window in the parent window's child list.
- See also: *TWindow::Next*
- ReceiveMessage** `LRESULT ReceiveMessage(UINT msg, WPARAM wParam = 0, LPARAM lParam = 0);`
- Called from *StdWndProc*, *ReceiveMessage* is the first member function called when a message is received. It calls *HandleMessage* from within the **try** block of the exception-handling code. In this way, exceptions can be caught and suspended before control is returned to exception-unsafe Window code.
- See also: *TWindow::HandleMessage*
- RedrawWindow** `inline BOOL RedrawWindow(TRect* update, HRGN hUpdateRgn,
 UINT redrawFlags = RDW_INVALIDATE | RDW_UPDATENOW
 | RDW_ERASE);`
- Calls the Windows API function *RedrawWindow* to redraw the rectangle at the address specified by *hUpdateRgn*. *redrawFlags* can be a combination of the Windows API *RDW_xxxx* constants. See the Windows API online Help for a list of these values.
- See also: *TWindow::GetUpdateRect*
- Register** `virtual BOOL Register();`
- Registers the Windows registration class of **this**, if **this** is not already registered. Calls *GetClassName* and *GetWindowClass* to retrieve the Windows registration class name and attributes of **this**. *Register* returns TRUE if **this** is registered.
- See also: *TWindow::GetClassName*, *TWindow::GetWindowClass*
- RegisterHotKey** `inline BOOL RegisterHotKey(int idHotKey, UINT modifiers, UINT virtKey);`



Calls the Windows API function *RegisterHotKey* to register a hotkey ID with the current application. *modifiers* can be a combination of keys that must be pressed to activate the specified hotkey, for example, `HOTKEYF_SHIFT`, `HOTKEYF_CONTROL`, and `HOTKEYF_ALT`.

See also: `::RegisterHotKey`, `TWindow::UnRegisterHotKey`

RemoveProp

```
HANDLE RemoveProp(WORD atom) const;
```

Calls the Windows API function *RemoveProp* to remove the property specified by *atom* from the application's property list.

See also: `TWindow::GetProp`

RemoveProp

```
HANDLE RemoveProp(const char far* str) const;
```

Calls the Windows API function *RemoveProp* to remove the property specified by *str* from the application's property list.

See also: `TWindow::GetProp`

ScreenToClient

```
inline void ScreenToClient(TPoint& point) const;
```

Calls the Windows API function *ScreenToClient* to use the screen coordinates specified in *point* to calculate client window's coordinates and then place the new coordinates into *point*.

See also: `::ScreenToClient`

ScrollWindow

```
inline void ScrollWindow(int dx, int dy, const TRect* scroll = 0,
                        const TRect far *clip = 0);
```

Calls the Windows API function *ScrollWindow* to scroll a window in the vertical (*dx*) and horizontal (*dy*) directions. *TRect* indicates the area to be scrolled. If 0, the entire client area is scrolled. *clip* specifies the clipping rectangle to be scrolled. Only the area within *clip* is scrolled. If *clip* is 0, the entire window is scrolled.

See also: `TWindow::ScrollWindowEx`, `::ScrollWindow`

ScrollWindowEx

```
inline void ScrollWindowEx(int dx, int dy, const TRect *scroll = 0,
                          const TRect far *clip = 0, HRGN hUpdateRgn = 0,
                          TRect far *update = 0, UINT flags = 0);
```

Calls the Windows API function *ScrollWindowEx* to scroll a window in the vertical (*dx*) and horizontal (*dy*) directions. *TRect* indicates the area to be scrolled. If 0, the entire client area is scrolled. *clip* specifies the clipping rectangle to be scrolled. Only the area within *clip* is scrolled. If *clip* is 0, the entire window is scrolled. *update* indicates the region that will receive the boundaries of the area that becomes invalidated as a result of scrolling.

flags, which determines how the window's children are scrolled, can be one of the Windows API *SW_xxxx* constants; for example, *SW_SCROLLCHILDREN*. See the Windows API online Help for a list of these constants.

See also: *TWindow::ScrollWindow*, *::ScrollWindowEx*, *TWindow::Show*

SendDlgItemMessage inline LRESULT SendDlgItemMessage(int childId, UINT msg,
WPARAM wParam = 0, LPARAM lParam = 0)
const;

Calls the Windows API function *SendDlgItemMessage* to send a message (*msg*) to the control specified in *childId*.

See also: *TWindow::SendMessage*, *::SendDlgItemMessage*

SendMessage inline LRESULT SendMessage(UINT msg, WPARAM wParam = 0, LPARAM lParam = 0)
const;

Wrapper function for the Window's API function by the same name, *SendMessage* sends a message (*msg*) to a specified window or windows. After it calls the window procedure, it waits until the window procedure has processed the message before returning.

See also: *TWindow::ChildBroadcastMessage*, *TWindow::HandleMessage*, *TWindow::SendDlgItemMessage*, *::SendMessage*

SendNotification inline void SendNotification(int id, int notifyCode, HWND hCtl,
UINT msg = WM_COMMAND);

Repacks a command message (*msg*) so that a child window (*hCtl*) can send a message to its parent regardless of whether this is a WIN16 or WIN32 application.

SetActiveWindow inline HWND SetActiveWindow();

Calls the Windows API function *SetActiveWindow* to activate a top-level window. Returns a handle to the previously active window.

See also: *TWindow::GetActiveWindow*, *::SetActiveWindow*

SetBkgndColor inline void SetBkgndColor(DWORD color);

Sets the background color (*color*) for the window. You can also get the current color of an element displayed on the screen. For example,

```
layout -> SetBkgndColor(GetSysColor(COLOR_APPWORKSPACE));
```

uses one of the Windows *COLOR* values (in this case, the color of multiple document interface applications (MDI). See the online Help file for a list of these display values.

See also: *::GetSysColor*

SetCaption

```
void SetCaption(const char far* title);
```

Copies *title* to an allocated string pointed to by *Title*. Sets the caption of the interface element to *title*. Deletes any previous title.

See also: *TWindow::GetWindowTitle*, *TWindow::Title*

SetCaretBlinkTime

```
inline static void SetCaretBlinkTime(WORD milliSecs);
```

Calls the Windows API function *SetCaretBlinkTime* to set the caret blink rate in milliseconds.

See also: *TWindow::GetCaretBlinkTime*, *::SetCaretBlinkTime*

SetCaretPos

```
inline static void SetCaretPos(int x, int y);
```

Calls the Windows API function *SetCaretPos* to set the position of the caret in the coordinates of the client window. *x* and *y* indicate the screen coordinates of the structure that receives the client coordinates of the caret.

See also: *TWindow::GetCaretPos*, *TWindow::ShowCaret*

SetCaretPos

```
inline static void SetCaretPos(const TPoint &pos);
```

Calls the Windows API function *SetCaretPos* to set the position of the caret in the coordinates of the client window. *TPoint* indicates the structure that receives the client coordinates of the caret.

See also: *TWindow::GetCaretPos*, *::SetCaretPos*, *TWindow::ShowCaret*

SetClassLong

```
inline long SetClassLong(int index) const;
```

Calls the Windows API function *SetClassLong* to set the long value at the specified offset (*index*). Depending on the value of *index*, *SetClassLong* sets a handle to a background brush, cursor, icon, menu, or function.

See also: *TWindow::GetClassLong*, *::SetClassLong*

SetClassWord

```
inline WORD SetClassWord(int index) const;
```

Calls the Windows API function *SetClassWord* to set the word value at the specified offset (*index*). Depending on the value of *index*, *SetClassLong* sets the number of bytes of class, window, or style information.

See also: *TWindow::GetClassWord*, *::SetClassWord*

SetCursor

```
inline BOOL SetCursor(TModule* module, TResId resId);
```

Sets the cursor position for the window using the given *inst* and *ResId*. If the mouse is over the client area, *SetCursor* changes the cursor that's displayed.

The following program shows how to create a window with a custom cursor:

```
class TCursorApp : public TApplication {
public:
    TCursorApp() : TApplication("CursorApp") {}
    void InitMainWindow() {
        MainWindow = new TFrameWindow (0, "Window with a Custom Cursor");
        MainWindow->SetCursor(IDL_APPLICATION, 100);
        MainWindow->SetIcon(IDL_APPLICATION, 100);
    }
};
```

See also: *TWindow::GetCursorPos*

SetDlgItem

```
void SetDlgItemInt(int childId, UINT value, BOOL isSigned = TRUE) const;
```

Calls the Windows API function *::SetDlgItem* to set the child window with the Id (*childId*) in the window to the integer value specified in *value*. If *isSigned* is TRUE, the value is signed.

See also: *TWindow::GetDlgItem, ::GetDlgItem*

SetDlgItemText

```
void SetDlgItemText(int childId, const char far* text) const;
```

Calls the Windows API function *SetDlgItemText* to set the text of a child with Id. *text* points to the text buffer containing the window caption or text that is to be copied into the child window caption or text.

See also: *TWindow::GetDlgItemText, ::SetDlgItemText*

SetDocTitle

```
virtual BOOL SetDocTitle(LPCSTR docname, int index);
```

Stores the title of the document.

SetFlag

```
inline void SetFlag(TWindowFlag mask);
```

If TRUE is supplied, the bits in *Attr.Flags* in *Mask* are set. Otherwise, the bit is cleared. *Mask* can be any one, or a combination, of the *wfXxxx* constants.

See also: *TWindow::IsFlagSet*

SetModule

```
inline void SetModule(TModule* module);
```

Sets the default module ID for this window.

See also: *TWindow::GetModule*

SetMenu

```
inline virtual BOOL SetMenu(HMENU);
```

TWindow's implementation calls the Windows API function *SetMenu* to set the specified window's menu to the menu indicated by *hMenu*. If *hMenu* is

0, the window's current menu is removed. *SetMenu* returns 0 if the menu remains unchanged; otherwise, it returns a nonzero value.

See also: *TWindow::GetMenu*, *TMDIFrame::SetMenu*, *::SetMenu*

SetParent

```
virtual void SetParent(TWindow* newParent);
```

Sets *Parent* to the specified parent window object. Removes **this** from the child list of the previous parent window, if any, and adds **this** to the new parent's child list.

See also: *TWindow::GetParent*

SetProp

```
inline BOOL SetProp(WORD atom, HANDLE data) const;
```

Calls the Windows API function *SetProp* to add an item to the property list of the specified window. *atom* contains a value that identifies the data entry to be added to the property list.

See also: *TWindow::GetProp*, *::SetProp*

SetProp

```
inline BOOL SetProp(const char far* str, HANDLE data) const;
```

Calls the Windows API function *SetProp* to add an item to the property list of the specified window. *str* points to the string used to identify the entry data to be added to the property list.

See also: *TWindow::GetProp*, *::SetProp*

SetScrollPos

```
inline int SetScrollPos(int bar, int pos, BOOL redraw = TRUE);
```

Calls the Windows API function *SetScrollPos* to set the thumb position in the scroll bar. *bar* identifies the position (horizontal, vertical, or scroll bar control) to return and can be one of the Windows API *SB_xxxx* constants.

See also: *TWindow::GetScrollPos*, *::SetScrollPos*

SetScrollRange

```
inline void SetScrollRange(int bar, int minPos, int maxPos,
                           BOOL redraw = TRUE);
```

Calls the Windows API function *SetScrollRange* to set the thumb position in the scroll bar. *bar* identifies the position (horizontal, vertical, or scroll bar control) to set and can be one of the Windows API *SB_xxxx* constants. *minPos* and *maxPos* specify the lower and upper range, respectively, of the scroll bar positions.

See also: *TWindow::GetScrollRange*, *::SetScrollRange*

SetSysModalWindow

```
inline HWND SetSysModalWindow();
```



Makes the indicated window a system-modal window.

See also: *TWindow::GetSysModalWindow*

- SetTimer** `inline UINT SetTimer(UINT timerId, UINT timeout, TIMERPROC proc = 0);`
- Calls the Windows API function *SetTimer*, which creates an event to be timed and enters any WM_TIMER messages into the message queue. *timerId* contains the ID number of the timer event to be entered, *timeout* specifies the length of time in milliseconds, and *proc* identifies the address of the function that's to be notified when the timed event occurs. If *proc* is 0, WM_TIMER messages are placed in the queue of the application that called *SetTimer* for this window.
- See also: *TWindow::KillTimer*, *::SetTimer*
- SetTransferBuffer** `void SetTransferBuffer(void* transferBuffer);`
- Sets *TransferBuffer* to *transferBuffer*.
- See also: *TWindow::Transfer*, *TWindow::TransferData*
- SetWindowLong** `long SetWindowLong(int index, long newLong);`
- Calls the Windows API function *SetWindowLong* to set the long value of *Id* in the *WindowsAttr* structure. Depending on the value of *index*, *SetWindowLong* sets a handle to a background brush, cursor, icon, module, menu, or window function. See the Windows API online Help for information about the GWL_XXXX values that represent styles.
- See also: *TWindow::GetWindowLong*, *::SetWindowLong*, *TWindow::SetWindowWord*
- SetWindowPlacement** `inline BOOL SetWindowPlacement(WINDOWPLACEMENT* place);`
- Calls the Windows API function *SetWindowPlacement* to set the window to the position indicated in *place*.
- See also: *TWindow::GetWindowPlacement*, *TWindow::Show*
- SetWindowPos** `inline void SetWindowPos(HWND hWndInsertAfter, const TRect &rect,
UINT flags);`
- Calls the Windows API function *SetWindowPos* to change the size of the window pointed to by *rect*. *flags* contains one of the SWP_XXXX constants that specify the size and position of the window. If *flags* is set to SWP_NOZORDER, *SetWindowPos* ignores the *hWndInsertAfter* parameter and retains the current ordering of the child, pop-up, or top-level windows.
- See also: *::SetWindowPos*
- SetWindowPos** `inline void SetWindowPos(HWND hWndInsertAfter, int x, int y, int w, int h,
UINT flags);`

Calls the Windows API function *SetWindowPos* to change the size of the window pointed to by *x*, *y*, *w*, and *h*. *flags* contains one of the *SWP_XXXX* constants that specify the size and position of the window. If *flags* is set to *SWP_NOZORDER*, *SetWindowPos* ignores the *hWndInsertAfter* parameter and retains the current ordering of the child, pop-up, or top-level windows.

See also: *::SetWindowPos*

SetWindowText

```
inline void SetWindowText(const char far* str);
```

Calls the Windows API function *SetWindowText*, which sets the window's title to a buffer pointed to by *str*. *maxCount* indicates the number of characters to copy into the buffer.

See also: *TWindow::GetWindowText*, *::SetWindowText*

SetWindowWord

```
WORD SetWindowWord(int index, WORD newWord);
```



Calls the Windows API function *SetWindowWord*, which changes information about the window. *index* specifies a byte offset of the word to be changed to the new value (*newWord*).

See also: *TWindow::GetWindowWord*, *TWindow::SetWindowLong*

Show

```
void Show(int cmdShow);
```

After ensuring that the *TWindow* interface element has a valid handle, *Show* displays the *TWindow* on the screen in a manner specified by *cmdShow*, which can be one of the following values:

Value	Description
SW_SHOWDEFAULT	Show the window in its default configuration. Should be used at startup.
SW_HIDE	Hide the window and activate another window.
SW_MINIMIZE	Minimize the window and activate the top-level window in the list.
SW_RESTORE	Same as SW_SHOWNORMAL.
SW_SHOW	Show the window in the window's current size and position.
SW_SHOWMAXIMIZED	Activate and maximize the window.
SW_SHOWMINIMIZED	Activate window as an icon.
SW_SHOWNA	Display the window as it is currently.
SW_SHOWMINNOACTIVE	Display the window as an icon.
SW_SHOWNORMAL	Activate and display the window in its original size and position.
SW_SHOWSMOOTH	Show the window after updating it in a bitmap.

ShowCaret

```
inline void ShowCaret();
```

Calls the Windows API function *::ShowCaret* to display the caret in the specified shape in the active window at the current position.

See also: *TWindow::CreateCaret*, *::ShowCaret*, *TWindow::HideCaret*

ShowOwnedPopups void ShowOwnedPopups(BOOL show);

Displays the popup window according to the value of *show*. See the function *Show* for a description of the SW_XXXX constants passed in *show*.

See also: *TWindow::Show*

ShowScrollBar inline void ShowScrollBar(int bar, BOOL show = TRUE);

Calls the Windows API function *ShowScrollBar* to display or hide the scroll bar. *bar* specifies whether the bar is a control or part of the window's nonclient area. If *bar* is part of the nonclient area, it can be one of the Windows API SB_XXXX constants (specifically, SB_BOTH, SB_CTRL, SB_HORZ, or SB_VERT). If *show* is TRUE, the scroll bar is displayed; if FALSE, it is hidden.

See also: *TWindow::GetScrollRange*, *::ShowScrollBar*

ShowWindow BOOL ShowWindow(int cmdShow);

Displays the window according to the value of *cmdShow*. See the function *Show* for a description of the SW_XXXX constants passed in *cmdShow*. *Show* is the preferred method of showing the window.

See also: *TWindow::Show*

SubclassWindowFunction void SubclassWindowFunction();

Installs the instance thunk as the *WindowProc* and saves the old window function in *DefaultProc*.

See also: *TWindow::DefaultProc*, *TWindow::DefaultProcessing*

Transfer virtual UINT Transfer(void* buffer, TTransferDirection direction);

Transfers data to or from any window with children and returns the total size of the buffer. *Transfer* is a general mechanism for data transfer that can be used with or without using *TransferData*(). The *direction* supplied specifies whether data is to be read from or written to the supplied buffer, or whether the size of the transfer data is simply to be returned. Data is not transferred between any child windows whose wfTransfer flag is not set. If data is transferred, the return value is the size (in bytes) of the transfer data.

See also: *TWindow::TransferData*

TransferData virtual void TransferData(TTransferDirection direction);

A window usually calls on *TransferData* during setup and closing of windows and relies on the constructor to set *TransferBuffer* to something meaningful. *TransferData* calls the *Transfer* member function of each

participating child window, passing a pointer to *TransferBuffer* as well as the direction specified in *direction* (*tdSetData*, *tdGetData*, or *tdSizeData*).

See also: *TWindow::EnableTransfer*, *TWindow::DisableTransfer*, *TWindow::SetupWindow*

UnregisterHotKey

```
inline BOOL UnregisterHotKey(int idHotKey);
```



Calls the Windows API function *UnregisterHotKey* to unregister a hotkey ID with the current application.

See also: *::UnRegisterHotKey*, *TWindow::RegisterHotKey*

UpdateWindow

```
inline void UpdateWindow();
```

Calls the Windows API function *UpdateWindow* to update the client area of the specified window by immediately sending a *WM_PAINT* message.

Validate

```
inline void Validate();
```

Calls the Windows API function *ValidateRect* to validate (that is, remove from the area to be updated) the entire client area.

See also: *TWindow::InValidate*

ValidateRect

```
inline void ValidateRect(const TRect&);
```

Calls the Windows API function *ValidateRect* to validate a portion of the client area indicated by *rect*.

See also: *TWindow::InvalidateRect*

ValidateRgn

```
inline void ValidateRgn(HRGN hRgn);
```

Calls the Windows API function *ValidateRgn* to validate the region of the window indicated by *hRgn*.

See also: *TWindow::InvalidateRgn*

WindowFromPoint

```
inline static HWND WindowFromPoint(const TPoint& point) const;
```

Returns a value indicating the window in which the specified point (*point*) exists.

See also: *TWindow::ChildWindowFromPoint*

WindowProc

```
virtual LRESULT WindowProc(UINT msg, WPARAM wParam, LPARAM lParam);
```

WindowProc processes incoming messages by calling *EvCommand* to handle *WM_COMMAND* messages and calling *EvCommandEnable* to handle *WM_COMMAND_ENABLE* messages. *EvCommandEnable* then checks to see if there is a response table entry to handle the event.

See also: *TWindow::DefWindowProc*

WinHelp

```
BOOL WinHelp(const char far* helpFile, UINT command, DWORD data);
```

Calls the Windows API function *WinHelp* to invoke a specified help system. *helpFile* points to a string containing the directory path and name of the help file. *command*, which indicates the type of help requested, can be one of the Windows `Help_XXXX` constants such as `HELP_CONTEXT`, `HELP_HELPONHELP`, `HELP_INDEX`, `HELP_MULTIKEY`, `HELP_QUIT`, or `HELP_SETINDEX`. *data* contains keywords that indicate the help topic items. For example, in the sample `ObjectWindows` file, `HELP.CPP`, *WinHelp* is called with the arguments `HELP_CONTEXT` and `HELP_MENUITEMA` if the F1 key is pressed.

```
void TOWlHelpWnd::CmMenuItemA()
{
    if (F1Pressed) {
        WinHelp(HelpFile, HELP_CONTEXT, HELP_MENUITEMA);
        F1Pressed = FALSE;
    } else {
        MessageBox("In Menu Item A command", Title, MB_ICONINFORMATION);
    }
}
```

You can also include bitmaps in your Help file by referencing their file names or by copying them from the Clipboard. For more information about how to create Help files, see the online Help documentation.

See also: `::WinHelp`

Protected data members

CursorInstance

```
TInstance* CursorInstance;
```

Holds the library ID for the specified cursor. A cursor ID of 0 indicates the standard system cursor.

CursorResId

```
TResId CursorResId;
```

Holds the cursor resource ID for the specified cursor. Could be one of the Windows `IDC_XXXX` constants if *CursorInstance* is 0. See the Windows API online Help for more information about `IDC_XXXX` constants.

hAccel

```
HACCEL hAccel;
```

Holds the handle to the current Windows accelerator table.

HCursor

```
HCURSOR HCursor;
```

Holds a handle to the window's cursor.

TransferBuffer

```
void* TransferBuffer;
```

TransferBuffer points to a buffer to be used in transferring data in and out of the *TWindow*. A *TWindow* assumes that the buffer contains data used by the windows in its child list. If 0, no data is to be transferred. See *tdxxxx* constants for a description of the transfer flags.

See also: *tdxxxx* constants

Protected member functions

CleanupWindow

```
virtual void CleanupWindow();
```

Called immediately before the HWindow becomes invalid, *CleanupWindow* gives derived classes an opportunity to clean up HWND related resources.

See also: *TWindow::SetupWindow*

DispatchScroll

```
void DispatchScroll(UINT scrollCode, UINT thumbPos, HWND hWndCtrl);
```

Called by *EvHScroll* and *EvVScroll* to dispatch messages from scroll bars.

See also: *TWindow::EvHScroll*, *TWindow::EvVScroll*

GetClassName

```
virtual char far* GetClassName();
```

Returns the Windows registration class name. The default class name is generated using "window" plus the module name.

See also: *TWindow::GetWindowClass*

GetWindowClass

```
virtual void GetWindowClass(WNDCLASS &wndClass);
```

Redefined by derived classes to fill the supplied Windows registration class structure with registration attributes. This function, along with *GetClassName*, allows Windows classes to be used for the specified ObjectWindows class and its derivatives.

See also: *TWindow::GetClassName*

LoadAcceleratorTable

```
void LoadAcceleratorTable();
```

Loads a handle to the window's accelerator table specified in *TWindowsAttr* structure (*Attr.AccelTable*). If no handle exists, then emits an "Unable to load accelerator table" diagnostic message.

See also: *TWindowsAttr* structure

RemoveChild

```
void RemoveChild(TWindow *);
```

Removes a child window. This family of ObjectWindows *TWindow* functions uses the ObjectWindows list of objects rather the Window's HWND list.

See also: *TWindow::GetFirstChild*, *TWindow::GetLastChild*, *TWindow::Next*, *TWindow::Previous*

SetupWindow

```
virtual void SetupWindow();
```

SetupWindow is the first virtual function called when the HWindow becomes valid. *TWindow*'s implementation performs window setup following the creation of an associated interface element. Iterates through the child list, attempting to create an associated interface element for each child window object for whom autocreation is enabled. (By default, autocreation is enabled for windows and controls, and disabled for dialog boxes.) *SetupWindow* then calls *TransferData*.

See *ObjectWindows Programmer's Guide* for more information about using *SetupWindow*.

SetupWindow can be redefined in derived classes to perform additional special initialization. Note that the HWindow is valid when the overridden *SetupWindow* is called and that the children's HWindows are valid after calling the base classes' *SetupWindow* function.

See also: *TFrameWindow::SetupWindow*, *TComboBox::SetupWindow*, *TWindow::CleanupWindow*

Response table entries

Response table entry	Member function
EV_WM_CREATE	EvCreate
EV_WM_CLOSE	EvClose
EV_WM_DESTROY	EvDestroy
EV_WM_SIZE	EvSize
EV_WM_MOVE	EvMove
EV_WM_NCDESTROY	EvNcDestroy
EV_WM_QUERYENDSESSION	EvQueryEndSession
EV_WM_COMPAREITEM	EvCompareItem
EV_WM_DELETEITEM	EvDeleteItem
EV_WM_DRAWITEM	EvDrawItem
EV_WM_MEASUREITEM	EvMeasureItem
EV_WM_CHILDINVALID	EvChildInvalid
EV_WM_VSCROLL	EvVScroll
EV_WM_HSCROLL	EvHScroll
EV_WM_PAINT	EvPaint
EV_WM_SETCURSOR	EvSetCursor
EV_WM_LBUTTONDOWN	EvLButtonDown
EV_COMMAND(CM_EXIT, CmExit)	CmExit

Response table entry	Member function
EV_WM_SYSCOLORCHANGE	EvSysColorChange
EV_WM_KILLFOCUS	EvKillFocus
EV_WM_ERASEBKGD	EvEraseBkgnd
EV_MESSAGE(WM_CTLCOLOMSGBOX, EvWin32CtlColor)	EvWin32CtlColor*
EV_MESSAGE(WM_CTLCOLOREDIT, EvWin32CtlColor)	EvWin32CtlColor*
EV_MESSAGE(WM_CTLCOLORLISTBOX, EvWin32CtlColor)	EvWin32CtlColor*
EV_MESSAGE(WM_CTLCOLORBTN, EvWin32CtlColor)	EvWin32CtlColor*
EV_MESSAGE(WM_CTLCOLORDLG, EvWin32CtlColor)	EvWin32CtlColor*
EV_MESSAGE(WM_CTLCOLORSCROLLBAR, EvWin32CtlColor)	EvWin32CtlColor*
EV_MESSAGE(WM_CTLCOLORSTATIC, EvWin32CtlColor)	EvWin32CtlColor*

* These response table entries are available only under the Win32 API.

TWindow::TXWindow class

window.h

A nested class, *TXWindow* describes an exception that results from trying to create an invalid window.

Public constructors

Constructor

```
TXWindow(TWindow* win = 0, UINT resourceId = IDS_INVALIDWINDOW);
```

Constructs a *TXWindow* object with a default resource ID of `IDS_INVALIDWINDOW`.

Public data members

Window

```
TWindow* Window;
```

Points to the window object that is associated with the exception.

Public member functions

Msg

```
static string Msg(TWindow*, UINT resourceid);
```

Converts the resource ID to a string and returns the string message.

TWindowFlag enum**window.h**

enum TWindowFlag

wfXxxx constants define bit masks for the internally used flag attributes of *TWindow*. A wfXxxx mask is defined for each of these attributes:

Table 1.1
TWindow's attribute
masks

Constant	Meaning if set
wfAlias	The window object is an alias for an existing HWND.
wfAutoCreate	Create window when parent window is created.
wfFromResource	Window is created from a resource definition.
wfMainWindow	Window is a main window.
wfPredefinedClass	Has a predefined Window's class.
wfShrinkToClient	Tells a frame window to shrink itself to fit around the client window.
wfTransfer	Participates in the <i>Transfer</i> mechanism.
wfStreamTop	Indicates the topmost window of the collection of windows to be streamed.

See also: *TWindow::CreateChildren*, *TWindow::EnableAutoCreate*

TWindowAttr struct**window.h**

A *TWindowAttr* is used to hold a *TWindow*'s attributes that are set during construction. Your program controls a window's creation by passing these values to one of *TWindow*'s creation routines. If the window is streamed, these attributes are also used for re-creation.

Public data members**AccelTable**

TResID AccelTable;

AccelTable holds the resource ID for the window's accelerator table.

See also: *TApplication::HAccTable*, *TWindow::LoadAcceleratorTable*

ExStyle

DWORD ExStyle;

ExStyle contains the extended style values of your window. These can be any one of the Windows API extended style constants (WS_EX_DLGMODALFRAME, WS_EX_NOPARENTNOTIFY, WS_EX_TOPMOST, WS_EX_SHADOW). See the Windows API online help for more information about these constants.

Id

int Id;

Id contains the identifier of the child window; for a dialog box control *Id* is its resource identifier. If `Win32` is defined, *Id* is set to `GetWindowLong` else *Id* is set to `GetWindowWord`.

Menu `TResID Menu;`

Menu contains the resource identifier that is the name of the window's menu if one is associated with the window.

Param `char far* Param;`

Param contains a value that is passed to Windows when the window is created. This value identifies a data block that is then available in the message response functions associated with `WM_CREATE`. *Param* is used by `TMDIClient` and can be useful when converting non-ObjectWindows code.

See also: `TWindow::EvCreate`

Style `DWORD Style;`

Style contains the values that define the style, shape, and size of your window. Although `TWindow` sets *Attr.Style* to `WS_CHILD` and `WS_VISIBLE`, you can also use other combinations of the Windows API style constants. See the Windows API online help for a description of the `WS_xxxx` window creation style constants.

X, Y, W, H `int X, Y, W, H;`

X and *Y* contain the screen coordinates of the top left corner of the window. *W* and *H* contain the width and height values of the window.

TWindowDC class

dc.h

Derived from `TDC`, `TWindowDC` is a DC class that provides access to the entire area owned by a window.

Public constructors and destructor

Constructor `TWindowDC(HWND wnd);`

Creates a `TWindow` object with the given owned window. The data member *Wnd* is set to *wnd*.

See also: `TWindowDC::Wnd`

Destructor `~TWindowDC();`

Destroys this object.

Protected constructors

Constructor

```
TWindowDC();
```

Used for derived classes only.

Protected data member

Wnd

```
HWND Wnd;
```

Holds a handle to the window owned by this DC.

See also: *TWindowDC::TWindowDC*

TWindowView class

docview.h

Derived from both *TWindow* and *TView*, *TWindowView* is a streamable base class that can be used for deriving window-based views. *TWindowView*'s functions override *TView*'s virtual function to provide their own implementation. By deriving a window-view class from *TWindow* and *TView*, you add window functionality to the view of your document.

Public constructors and destructor

Constructor

```
TWindowView (TDocument& doc, TWindow* parent = 0);
```

Constructs a *TWindowView* interface object associated with the window view. Sets *ViewId* to *NextViewId*. Calls the associated document's *AttachView* to attach the view to the document.

Destructor

```
inline ~TWindowView()
```

Destroys a *TWindowView* object and calls the associated document's *DetachView* function to detach the view from the associated document.

Public member functions

CanClose

```
inline BOOL CanClose();
```

Overrides *TView::CanClose* and returns nonzero if the window can be closed. Checks all of the associated document's *CanClose* functions. These must return nonzero before the window view can be closed.

See also: *TView::CanClose*, *TWindow::CanClose*

GetViewName

```
inline LPCSTR GetViewName();
```

Overrides *TView::GetViewName* and returns *StaticName*, the name of the view.

See also: *TView::GetViewName*

GetWindow

```
TWindow* GetWindow()
```

Overrides *TView::GetWindow* and returns the *TWindowView* object as a *TWindow*.

See also: *TEditView::GetWindow*, *TView::GetWindow*

SetDocTitle

```
inline BOOL SetDocTitle(LPCSTR docname, int index)
```

Overrides *TView::SetDocTitle* and stores the document title. This name is forwarded up the parent chain until a *TFrameWindow* object accepts the data and displays it in its caption.

See also: *TView::SetDocTitle*, *TWindow::SetDocTitle*

StaticName

```
inline static LPCSTR StaticName();
```

Returns "Window View," the descriptive name of the view. This title is displayed in the user-interface box.

Response table entries

Response table entry	Member function
EV_VN_ISWINDOW	VnlsWindow

TXCompatibility class**except.h**

Describes an exception that results from setting *TModule::Status* to nonzero. This exception is included for backward compatibility with ObjectWindows 1.0.

Public constructors

Constructor

```
TXCompatibility(int statusCode);
```

Constructs a *TXCompatibility* object.

Public member functions

MapStatusCodeToString

```
static string MapStatusCodeToString(int statusCode);
```

Retrieves *Tmodule*'s status code and converts it to a string.

TXOwl class

except.h

TXOwl is a base class designed to describe exceptions, that is, abnormal conditions outside the program's control. In most cases, you will derive a new class from *TXOwl* instead of using this one directly.

Each of the exception classes describes a particular type of exception. When your program encounters a given situation that's likely to produce this exception, it passes control to the specified exception-handling object. If you use exceptions in your code, you can avoid having to scatter error-handling procedures throughout your program.

To create an exception handler, place the keyword **try** before the block of code that might produce the abnormal condition (the code that might generate an exception object) and the keyword **catch** before the block of code that follows the **try** block. If an exception is thrown within the **try** block, the classes within each of the subsequent **catch** clauses are checked in sequence. The first one that matches the class of the exception object is executed.

The following example from MDIFILE.CPP, a sample program on your distribution disk, shows how to set up a **try/catch** block around the code that might throw an exception.

```
void TMDIFileApp::CmRestoreState()
{
    char* errorMsg = 0;

    ifstream is(DskFile);
    if (is.bad())
        errorMsg = "Unable to open desktop file.";

    // try block of code //

    else {
```

```

        if (Client->CloseChildren()) {
            try {
                is >>* this;
                if (is.bad())
                    errorMsg = "Error reading desktop file.";
                else
                    Client->CreateChildren();
            }
        }
        // catch block of code //
        catch (xalloc) {
            Client->CloseChildren();
            errorMsg = "Not enough memory to open file.";
        }
    }
    if (errorMsg)
        MainWindow->MessageBox(errorMsg, "Error", MB_OK | MB_ICONEXCLAMATION);
}

```

See the *ObjectWindows Programmer's Guide* for more information about how to write programs using exception handlers.

Public constructors and destructor

Constructor

```
inline TXOwl(const string& msg) : xmsg(msg), ResId(0);
```

Constructs a *TXOwl* object with a string message (*msg*).

```
TXOwl(unsigned resId, HINSTANCE instance = 0);
```

Constructs a *TXOwl* object with a resource ID.

Destructor

```
inline virtual ~TXOwl();
```

Destroys a *TXOwl* object.

Public member functions

Clone

```
virtual TXOwl* Clone();
```

Makes a copy of the exception object. *Clone* must be implemented in any class derived from *TXOwl*.

ResourceIdToString

```
static string ResourceIdToString(BOOL* found, unsigned resId,
                                HINSTANCE instance = 0);
```

Converts the resource ID to a string and returns a string that identifies the exception. If the string message cannot be loaded, returns a “not found” message.

Throw `virtual void Throw();`

Throws the exception object. *Throw* must be implemented in any class derived from *TXOwl*.

Unhandled `virtual int Unhandled(TModule* app, unsigned promptResId);`

Unhandled is called when an unhandled exception is caught at the main message loop level.

Vnxxxx view notification constants

docview.h

The *view notification* constants are used to notify the view of a given event.

Table 1.36
Vnxxxx view
notification IDs

Constant	Meaning
<code>vnCommit</code>	Changes are committed to the document.
<code>vnCustomBase</code>	Base event for document notifications.
<code>vnDocOpened</code>	Document has been opened.
<code>vnDocClosed</code>	Document has been closed.
<code>vnIsDirty</code>	Is TRUE if uncommitted changes are present.
<code>vnIsWindow</code>	Is TRUE if the HWND passed belongs to this view.
<code>vnRevert</code>	Document's previous data is reloaded and overwrites the view's current data.
<code>vnViewOpened</code>	A new view has been constructed.
<code>vnViewClosed</code>	A view is about to be destroyed.

See also: *TListView*, *TEditView*

Voxxxx validator constants

validate.h

Validator constants represent bits in the bitmapped *Options* word in validator objects.

Table 1.37
Voxxxx validator
constants

Constant	Meaning
<code>voFill</code>	Used by picture validators to indicate whether to fill in literal characters as the user types.
<code>voTransfer</code>	The validator handles data transfer for the input line. Currently only used by range validators.
<code>voOnAppend</code>	Used by picture validators to determine how to interact with edit controls.
<code>voReserved</code>	The bits in this mask are reserved.

xs exception status enum

These bit flags define the types of exceptions that are caught and suspended. *TApplication*'s functions *SuspendThrow* and *QueryThrow* return the values of these bit flags.

Table 1.38
xs exception status
enum

Constant	Meaning
xsUnknown	Unknown exception
xsBadCast	<i>Bad_cast</i> exception
xsBadTypeid	<i>Bad_typeid</i> exception
xsMsg	Any exception derived from <i>xmsg</i>
xsAlloc	<i>xalloc</i> exception
xsOwl	<i>TXOwl</i> exception

See also: *TXOwl*, *TApplication::QueryThrow*, *TApplication::SuspendThrow*

Event handlers

This chapter includes several tables that list predefined `ObjectWindows` response-table macros and event-handling functions. Each table lists the name of the `ObjectWindows` macro, any required macro arguments, and the associated event-handling function. The tables are organized in the following manner:

Table	Description	Page
2.1	Command messages	458
2.2	Standard Windows messages	458
2.3	Child ID notification messages	461
2.4	Button notification messages	461
2.5	Combo box notification messages	461
2.6	List box notification messages	462
2.7	Edit control notification messages	462
2.8	Document manager messages	462
2.9	Document view messages	463
2.10	VBX Control messages	463

Some event-handling functions or messages have no predefined names. In these cases, the generic term *UserName* is used to indicate that you can use any function name you want to as long as the function's signature matches the signature required by the response table macro. Similarly, the term *UserMessage* indicates that you can define your own message ID.

This chapter uses the following conventions when listing the arguments of the message macros: ID refers to the child window's ID (for example, `ID_GROUPBOX`), CMD ID refers to the command ID (for example, `CM_FILENEW`), and Code refers to the notification code (for example, `BN_CLICKED`) that is being sent.

See Chapter 5 in the *ObjectWindows Programmer's Guide* for information about creating response table entries that associate these message macros with their corresponding event-handling functions.

The following macros handle WM_COMMAND messages:

Table 2.1: WM_COMMAND messages

Macro	Macro arguments	Response function declaration
EV_COMMAND	CMD ID, <i>UserName</i>	void <i>UserName</i> ()
EV_COMMAND_AND_ID	CMD ID, <i>UserName</i>	void <i>UserName</i> (WPARAM)
EV_COMMAND_ENABLE	CMD ID, <i>UserName</i>	void <i>UserName</i> (TCommandEnabler&)
EV_MESSAGE	<i>UserMessage</i> , <i>UserName</i>	LRESULT <i>UserName</i> (WPARAM,LPARAM)
EV_REGISTERED	Registered name, <i>UserName</i>	LRESULT <i>UserName</i> (WPARAM,LPARAM)

Table 2.2 lists the macros that handle Windows messages. To determine the name of the Windows message that corresponds to the EV_XXXX macro, remove the EV_ prefix. For example, WM_ACTIVATE is the name of the Windows message that the EV_WM_ACTIVATE macro handles. These macros, which crack the standard Windows messages (break the LPARAM and WPARAM parameters into separate parts), take no arguments. They pass the cracked parameters directly to the predefined EVxxxx message function. The standard Windows messages are described in your Windows documentation.

Table 2.2: WM_xxxx Window messages

Macro	Response function declaration
EV_WM_ACTIVATE	void EvActivate(UINT active, BOOL minimized, HWND hWndOther)
EV_WM_ACTIVATEAPP	void EvActivateApp(BOOL active, HANDLE threadId) ¹ void EvActivateApp(BOOL active, HTASK hTask) ²
EV_WM_ASKCBFORMATNAME	void EvAskCBFormatName(UINT bufLen, char far* buffer)
EV_WM_CANCELMODE	void EvCancelMode()
EV_WM_CHANGECHAIN	void EvChangeCBChain(UINT bufLen, char far* buffer)
EV_WM_CHAR	void EvChar(UINT key, UINT repeatCount, UINT flags)
EV_WM_CHARTOITEM	int EvCharToItem(UINT key, HWND hWndListBox, UINT caretPos)
EV_WM_CHILDACTIVATE	void EvChildActivate()
EV_WM_CHILDINVALID	void EvChildInvalid(HWND)
EV_WM_CLOSE	void EvClose()
EV_WM_COMPACTING	void EvCompacting(UINT compactRatio)
EV_WM_COMPAREITEM	LRESULT EvCompareItem(UINT ctrlId, COMPAREITEMSTRUCT far& compareInfo)
EV_WM_CREATE	int EvCreate(CREATESTRUCT far &)
EV_WM_CTLCOLOR	HBRUSH EvCtlColor(HDC, HWND hWndChild, UINT ctlType)
EV_WM_DEADCHAR	void EvDeadChar(UINT deadKey, UINT repeatCount, UINT flags)
EV_WM_DELETEITEM	void EvDeleteItem(UINT ctrlId, DELETEITEMSTRUCT far& deleteInfo)
EV_WM_DESTROY	void EvDestroy()
EV_WM_DESTROYCLIPBOARD	void EvDestroyClipboard()
EV_WM_DEVMODECHANGE	void EvDevModeChange(char far* devMode)
EV_WM_DRAWCLIPBOARD	void EvDrawClipboard()
EV_WM_DRAWITEM	void EvDrawItem(UINT ctrlId, DRAWITEMSTRUCT far& drawInfo)

Table 2.2: WM_XXXX Window messages (continued)

EV_WM_DROPFILES	void EvDropFiles(TDropInfo dropInfo)
EV_WM_ENABLE	void EvEnable(BOOL enabled)
EV_WM_ENDSESSION	void EvEndSession(BOOL endSession)
EV_WM_ENTERIDLE	void EvEnterIdle(UINT source, HWND hWndDlg)
EV_WM_ERASEBKGD	BOOL EvEraseBkgnd(HDC)
EV_WM_FONTCHANGE	void EvFontChange()
EV_WM_GETDLGCODE	UINT EvGetDlgCode()
EV_WM_GETMINMAXINFO	void EvGetMinMaxInfo(MINMAXINFO far &)
EV_WM_GETTEXT	void EvGetText(UINT bufLen, char far* buffer)
EV_WM_GETTEXTLENGTH	UINT EvGetTextLength()
EV_WM_HOTKEY	void EvHotKey(int idHotKey) ¹
EV_WM_HSCROLL	void EvHScroll(UINT scrollCode, UINT thumbPos, HWND hWndCtl)
EV_WM_HSCROLLCLIPBOARD	void EvHScrollClipboard(HWND hWndCBViewer, UINT scrollCode, UINT pos)
EV_WM_ICONERASEBKGD	void EvIconEraseBkgnd(HDC)
EV_WM_INITMENU	void EvInitMenu(HMENU)
EV_WM_INITMENUPOPUP	void EvInitMenuPopup(HMENU hPopupMenu, UINT index, BOOL sysMenu)
EV_WM_INPUTFOCUS	void EvInputFocus(BOOL gainingFocus) ¹
EV_WM_KEYDOWN	void EvKeyDown(UINT key, UINT repeatCount, UINT flags)
EV_WM_KEYUP	void EvKeyUp(UINT key, UINT repeatCount, UINT flags)
EV_WM_KILLFOCUS	void EvKillFocus(HWND hWndGetFocus)
EV_WM_LBUTTONDOWNBLCK	void EvLButtonDownBlck(UINT modKeys, TPoint& point)
EV_WM_LBUTTONDOWN	void EvLButtonDown(UINT modKeys, TPoint& point)
EV_WM_LBUTTONUP	void EvLButtonUp(UINT modKeys, TPoint& point)
EV_WM_MBUTTONDOWNBLCK	void EvMButtonDownBlck(UINT modKeys, TPoint& point)
EV_WM_MBUTTONDOWN	void EvMButtonDown(UINT modKeys, TPoint& point)
EV_WM_MBUTTONUP	void EvMButtonUp(UINT modKeys, TPoint& point)
EV_WM_MDIACTIVATE	void EvMDIActivate(HWND hWndActivated, HWND hWndDeactivated)
EV_WM_MDICREATE	LRESULT EvMDICreate(MDICREATESTRUCT far& createStruct)
EV_WM_MENUCHAR	LRESULT EvMenuChar(UINT nChar, UINT menuType)
EV_WM_MENUSELECT	void EvMenuSelect(UINT menuItemId, UINT flags)
EV_WM_MEASUREITEM	void EvMeasureItem(UINT ctrlId, MEASUREITEMSTRUCT far& measureInfo)
EV_WM_MOUSEACTIVATE	UINT EvMouseActivate(HWND hWndTopLevel, UINT hitTestCode, UINT msg)
EV_WM_MOUSEMOVE	void EvMouseMove(UINT modKeys, TPoint& point)
EV_WM_MOVE	void EvMove(TPoint &clientOrigin)
EV_WM_NCACTIVATE	BOOL EvNCActivate(BOOL active)
EV_WM_NCCALCSIZE	UINT EvNCCalcSize(BOOL calcValidRects, NCCALCSIZE_PARAMS far &)
EV_WM_NCCREATE	BOOL EvNCCreate(CREATESTRUCT far &)
EV_WM_NCDESTROY	void EvNCDestroy()
EV_WM_NCHITTEST	UINT EvNCHitTest(TPoint& point)
EV_WM_NCLBUTTONDOWNBLCK	void EvNCLButtonDownBlck(UINT hitTest, TPoint& point)
EV_WM_NCLBUTTONDOWN	void EvNCLButtonDown(UINT hitTest, TPoint& point)
EV_WM_NCLBUTTONUP	void EvNCLButtonUp(UINT hitTest, TPoint& point)
EV_WM_NCMBUTTONDBLCLK	void EvNCMButtonDbClk(UINT hitTest, TPoint& point)
EV_WM_NCMBUTTONDOWN	void EvNCMButtonDown(UINT hitTest, TPoint& point)
EV_WM_NCMBUTTONUP	void EvNCMButtonUp(UINT hitTest, TPoint& point)
EV_WM_NCMOUSEMOVE	void EvNCMouseMove(UINT hitTest, TPoint& point)
EV_WM_NCPAINT	void EvNCPaint()

Table 2.2: WM_xxxx Window messages (continued)

EV_WM_NCRBUTTONDBLCLK	void EvNCRButtonDbkClk(UINT hitTest, TPoint& point)
EV_WM_NCRBUTTONDOWN	void void EvNCRButtonDown(UINT hitTest, TPoint& point)
EV_WM_NCRBUTTONUP	void EvNCRButtonUp(UINT hitTest, TPoint& point)
EV_WM_OTHERWINDOWCREATED	void EvOtherWindowCreated(HWND hWndOther) ¹
EV_WM_OTHERWINDOWDESTROYED	void EvOtherWindowDestroyed(HWND hWndOther) ¹
EV_WM_PAINT	void EvPaint()
EV_WM_PAINTCLIPBOARD	void EvPaintClipboard(HWND, HANDLE hPaintStruct)
EV_WM_PAINTICON	void EvPaintIcon() ¹
EV_WM_PALETTECHANGED	void EvPaletteChanged(HWND hWndPalChg)
EV_WM_PALETTEISCHANGING	void EvPalettelsChanging(HWND hWndPalChg)
EV_WM_PARENTNOTIFY	void EvParentNotify(UINT event, UINT childHandleOrX, UINT childIDOrY)
EV_WM_POWER	int EvPower(UINT powerEvent)
EV_WM_QUERYDRAGICON	HANDLE EvQueryDragIcon()
EV_WM_QUERYENDSESSION	BOOL EvQueryEndSession()
EV_WM_QUERYNEWPALETTE	BOOL EvQueryNewPalette()
EV_WM_QUERYOPEN	BOOL EvQueryOpen()
EV_WM_RBUTTONDBLCLK	void EvRButtonDbkClk(UINT modKeys, TPoint& point)
EV_WM_RBUTTONDOWN	void EvRButtonDown(UINT modKeys, TPoint& point)
EV_WM_RBUTTONUP	void EvRButtonUp(UINT modKeys, TPoint& point)
EV_WM_RENDERALLFORMATS	void EvRenderAllFormats()
EV_WM_RENDERFORMAT	void EvRenderFormat(UINT dataFormat)
EV_WM_SETCURSOR	BOOL EvSetCursor(HWND hWndCursor, UINT hitTest, UINT mouseMsg)
EV_WM_SETFOCUS	void EvSetFocus(HWND hWndLostFocus)
EV_WM_SETFONT	void EvSetFont(HFONT)
EV_WM_SETTEXT	void EvSetText(char far* text)
EV_WM_SHOWWINDOW	void EvShowWindow(BOOL show, UINT status)
EV_WM_SIZE	void EvSize(UINT sizeType, TSize& size)
EV_WM_SIZECLIPBOARD	void EvSizeClipboard(HWND hWndViewer, HANDLE hRect)
EV_WM_SPOOLERSTATUS	void EvSpoolerStatus(UINT jobStatus, UINT jobsLeft)
EV_WM_SYSCCHAR	void EvSysChar(UINT key, UINT repeatCount, UINT flags)
EV_WM_SYSCOLORCHANGE	void EvSysColorChange()
EV_WM_SYSCOMMAND	void EvSysCommand(UINT cmdType, TPoint& point)
EV_WM_SYSDEADCHAR	void EvSysDeadChar(UINT key, UINT repeatCount, UINT flags)
EV_WM_SYSKEYDOWN	void EvSysKeyDown(UINT key, UINT repeatCount, UINT flags)
EV_WM_SYSKEYUP	void EvSysKeyUp(UINT key, UINT repeatCount, UINT flags)
EV_WM_SYSTEMERROR	void EvSystemError(UINT error)
EV_WM_TIMECHANGE	void EvTimeChange()
EV_WM_TIMER	void EvTimer(UINT timerId)
EV_WM_VKEYTOITEM	int EvVKeyToItem(UINT key, HWND hWndListBox, UINT caretPos)
EV_WM_VSCROLL	void EvVScroll(UINT scrollCode, UINT thumbPos, HWND hWndCtl)
EV_WM_VSCROLLCLIPBOARD	void EvVScrollClipboard(HWND hWndCBViewer, UINT scrollCode, UINT pos)
EV_WM_WINDOWPOSCHANGED	void EvWindowPosChanged(WINDOWPOS far &>windowPos)
EV_WM_WINDOWPOSCHANGING	void EvWindowPosChanging(WINDOWPOS far &>windowPos)
EV_WM_WININICHANGE	void EvWinIniChange(char far* section)

¹ WIN32 API only

² WIN16 API only

The following macros handle messages that a child window sends to its parent:

Table 2.3: Child ID notification messages

Macro	Macro arguments	Response function declaration
EV_CHILD_NOTIFY_AND_CODE	ID, Code, <i>UserName</i>	void <i>UserName</i> (WPARAM)
EV_CHILD_NOTIFY_ALL_CODES	ID, <i>UserName</i>	void <i>UserName</i> (UINT)
EV_CHILD_NOTIFY_AT_CHILD	Code, <i>UserName</i>	void <i>UserName</i> ()
EV_CHILD_NOTIFY	ID, Code, <i>UserName</i>	void <i>UserName</i> ()

The following button macros handle BN_xxxx notification codes. To determine the name of the notification code that corresponds to the EV_XXXX macro, remove the EV_ prefix.

Table 2.4: Button notification messages

Macro	Macro arguments	Response function declaration
EV_BN_CLICKED	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_BN_DISABLE	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_BN_DOUBLECLICKED	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_BN_HILITE	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_BN_PAINT	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_BN_UNHILITE	ID, <i>UserName</i>	void <i>UserName</i> ()

The following combo box macros handle CBN_xxxx notification codes. To determine the name of the notification code that corresponds to the EV_XXXX macro, remove the EV_ prefix.

Table 2.5: Combo box notification messages

Macro	Macro arguments	Response function declaration
EV_CBN_CLOSEUP	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_CBN_DBLCLK	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_CBN_DROPDOWN	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_CBN_EDITCHANGE	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_CBN_EDITUPDATE	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_CBN_ERRSPACE	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_CBN_KILLFOCUS	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_CBN_SELCHANGE	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_CBN_SELENDCHANGE	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_CBN_SELENDOK	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_CBN_SETFOCUS	ID, <i>UserName</i>	void <i>UserName</i> ()

The following list box macros handle LBN_XXXX notification codes. To determine the name of the notification code that corresponds to the EV_XXXX macro, remove the EV_ prefix.

Table 2.6: List box notification messages

Macro	Macro arguments	Response function declaration
EV_LBN_DBLCLK	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_LBN_ERRSPACE	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_LBN_KILLFOCUS	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_LBN_SELCANCEL	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_LBN_SELCHANGE	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_LBN_SETFOCUS	ID, <i>UserName</i>	void <i>UserName</i> ()

The following edit control macros handle EN_XXXX notification codes. To determine the name of the notification code that corresponds to the EV_XXXX macro, remove the EV_ prefix.

Table 2.7: Edit control notification messages

Macro	Macro arguments	Response function declaration
EV_EN_CHANGE	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_EN_ERRSPACE	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_EN_HSCROLL	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_EN_KILLFOCUS	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_EN_MAXTEXT	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_EN_SETFOCUS	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_EN_UPDATE	ID, <i>UserName</i>	void <i>UserName</i> ()
EV_EN_VSCROLL	ID, <i>UserName</i>	void <i>UserName</i> ()

The following macros handle messages generated by the document manager:

Table 2.8: New document and view messages

Macro	Macro arguments	Response function declaration
EV_OWLDOCUMENT	ID, <i>UserName</i>	void <i>UserName</i> (TDocument& document)
EV_OWLNOTIFY	ID, <i>UserName</i>	BOOL <i>UserName</i> (LPARAM&)
EV_OWLVIEW	ID, <i>UserName</i>	void <i>UserName</i> (TView& view)

The following macros handle view-related messages generated by the document manager: See Chapter 9 in the *ObjectWindows Programmer's Guide* for more information about how to use these macros. *VnHandler* is a generic term for the view notification handler function.

Table 2.9: Document view messages

Macro	Response function declaration
EV_VN_VIEWOPENED	BOOL VnViewOpened(TView* view)
EV_VN_VIEWCLOSED	BOOL VnViewClosed(TView* view)
EV_VN_DOCOPENED	BOOL VnDocOpened(int openMode)
EV_VN_DOCCLOSED	BOOL VnDocClosed(int openMode)
EV_VN_COMMIT	BOOL VnCommit(BOOL force)
EV_VN_REVERT	BOOL VnRevert(BOOL clear)
EV_VN_ISDIRTY	BOOL VnIsDirty()
EV_VN_ISWINDOW	BOOL VnIsWindow(HWND hWnd)

The following macros handle WM_VBXFIREEVENT messages generated by VBX controls. *EvHandler* is a generic term for a specific VBX control message (such as *EvClick*).

Table 2.10: VBX messages

Macro	Macro arguments	Response function declaration
EV_VBXEVENTNAME	ID, event, <i>EvHandler</i>	void <i>EvHandler</i> (VBXEVENT FAR *event)
EV_VBXEVENTINDEX	ID, event, <i>EvHandler</i>	void <i>EvHandler</i> (VBXEVENT FAR *event)

Dispatch functions

See Chapter 5 in the *ObjectWindows Programmer's Guide* for more information about how to use dispatch functions.

This chapter alphabetically lists the ObjectWindows dispatch functions. Dispatch functions separate the lParam and wParam parameters of Windows messages into their respective data types and pass control to an ObjectWindows member function.

For example, when Windows sends an application a WM_CTLCOLOR message, the wParam is really an HDC, and the lParam has a HWND and a UINT hidden inside. After the dispatch function cracks the wParam and lParam into their constituent parts, it dispatches the Windows WM_CTLCOLOR message to the following ObjectWindows member function:

```
HBRUSH EvCtlColor(HDC, HWND, UINT);
```

Although dispatch functions are written for specific Windows API messages, they have no knowledge of the actual Windows messages they are sending. Instead, at run time, *TEventHandler::Dispatch* calls the appropriate dispatch function which then cracks the message and calls the appropriate member function, passing the response table's *pmf* (pointer to a member function) to the member function. These functions are never called directly.

The following four parameters are common to all dispatch functions.

- *GENERIC& generic* is the pointer to the object (for example, *TEdit*).
- *GENERIC::*pmf* is the pointer to the member function (for example, *EvActivate*).
- *WPARAM* is one of the message parameters the dispatch function cracks.
- *LPARAM* is one of the message parameters the dispatch function cracks.

The signature of the function used in the response table macro depends on the type of message cracking the function performs. The first letter of the signature indicates the return type (for example, *U* indicates an unsigned integer). The second group of letters signifies the arguments of the function (for example, *POINT* of type *TPOINT*, or *U* of type *UINT*). Note that the signature of the function serves as a template that enforces type checking.

The following table lists the abbreviations of the member functions' signatures and their corresponding data types.

Abbreviation	Data type
v	void
i	int
U	UINT
B	BOOL
H	HANDLE
W	HWND
S	LPSTR

ObjectWindows uses the same dispatch functions for messages that require the same type of cracking. For example, both the WM_HSCROLL and WM_VSCROLL event handlers have **void (*) (UINT, UINT, HWND)** as their signature. The Windows message also has the wParam and lParam parameters in the same place. Therefore, ObjectWindows uses the same dispatch functions for both of these messages. See Chapter 2 for a list of Window messages and the corresponding ObjectWindows event handlers.

HBRUSH_HDC_W_U_Dispatch

dispatch.h

Passes an HDC, an HWND, and a UINT and returns an HBRUSH. The HDC is wParam, the HWND is the LOWORD of lParam, and the UINT is lParam.

```
LRESULT HBRUSH_HDC_W_U_Dispatch(GENERIC& generic, HBRUSH
                                (GENERIC::*pmf) (HDC, HWND, UINT), WPARAM
                                wParam, LPARAM lParam);
```

i_LPARAM_Dispatch

dispatch.h

Passes LPARAM as the only parameter and returns an int.

```
LRESULT i_LPARAM_Dispatch(GENERIC& generic, int (GENERIC::*pmf) (LPARAM),
                          WPARAM wParam, LPARAM lParam);
```

i_U_W_U_Dispatch

dispatch.h

Passes a UINT, HWND, and UINT and returns an int.

```
LRESULT i_U_W_U_Dispatch(GENERIC& generic, int (GENERIC::*pmf)(UINT, HWND,
    UINT), WPARAM wParam, LPARAM lParam);
```

i_WPARAM_Dispatch

dispatch.h

Passes WPARAM as the only parameter and returns an **int**.

```
LRESULT i_WPARAM_Dispatch(GENERIC& generic, BOOL (GENERIC::*pmf)(WPARAM),
    WPARAM wParam, LPARAM lParam);
```

LRESULT_WPARAM_LPARAM_Dispatch

dispatch.h

This dispatcher performs no message cracking. Instead, it passes *wParam* and *lParam* and returns an *LRESULT*.

```
LRESULT LRESULT_WPARAM_LPARAM_Dispatch(GENERIC& generic, LRESULT
    (GENERIC::*pmf)(WPARAM, LPARAM),
    WPARAM wParam, LPARAM lParam);
```

U_POINT_Dispatch

dispatch.h

U_POINT passes a reference to **POINT** and returns a **UINT**. Under **WIN32**, **POINT** is **lParam**; under **WIN16**, **POINT** is a reference to **lParam**.

```
LRESULT U_POINT_Dispatch(GENERIC& generic, UINT (GENERIC::*pmf)(TPoint
    &), WPARAM wParam, LPARAM lParam);
```

U_U_U_U_Dispatch

dispatch.h

Passes three **UINTs** and returns a **UINT**. The first **UINT** is *wParam*, the second **UINT** is the **LOWORD** of *lParam*, and the third **UINT** is the **HIWORD** of *lParam*.

```
LRESULT U_U_U_U_Dispatch(GENERIC& generic,
    UINT (GENERIC::*pmf)(UINT, UINT, UINT),
    WPARAM wParam, LPARAM lParam);
```

U_U_U_W_Dispatch**dispatch.h**

Passes two UINTs and an HWND and returns a UINT. Under WIN32, the first UINT is the LOWORD of wParam, the second UINT is the HIWORD of wParam, the HWND is lParam. Under WIN16, the first UINT is wParam, the second UINT is the LOWORD of lParam, and the HWND is the HIWORD of lParam.

```
LRESULT U_U_U_W_Dispatch(GENERIC& generic,
                        UINT (GENERIC::*pmf) (UINT, UINT, HWND),
                        WPARAM wParam, LPARAM lParam);
```

U_Dispatch**dispatch.h**

Passes no arguments and returns a UINT.

```
LRESULT U_Dispatch(GENERIC& generic, UINT (GENERIC::*pmf) (),
                  WPARAM wParam, LPARAM lParam);
```

U_WPARAM_LPARAM_Dispatch**dispatch.h**

Passes wParam and lParam and returns a UINT.

```
LRESULT U_WPARAM_LPARAM_Dispatch(GENERIC& generic,
                                  UINT (GENERIC::*pmf) (WPARAM, LPARAM),
                                  WPARAM wParam, LPARAM lParam);
```

v_LPARAM_Dispatch**dispatch.h**

Passes lParam (ignores wParam) as the only parameter and always returns 0.

```
LRESULT v_LPARAM_Dispatch (GENERIC& generic,
                           void (GENERIC::*pmf) (LPARAM), WPARAM wParam,
                           LPARAM lParam);
```

v_POINT_Dispatch**dispatch.h**

Passes a reference to a POINT and always returns 0. Under WIN32, passes lParam; under WIN16, passes a reference to lParam.

```
LRESULT v_POINT_Dispatch (GENERIC& generic, void (GENERIC::*pmf) (TPoint
&), WPARAM wParam, LPARAM lParam);
```

v_POINTER_Dispatch

dispatch.h

Passes an LPARAM as a void* pointer and always returns 0.

```
LRESULT v_POINTER_Dispatch(GENERIC& generic, void (GENERIC::*pmf)(void*),
WPARAM wParam, LPARAM lParam);
```

v_U_B_W_Dispatch

dispatch.h

Passes a UINT, a BOOL, and an HWND and always returns 0. Under WIN32, the UINT is the LOWORD of wParam, the BOOL is the HIWORD of wParam, and the HWND is lParam. Under WIN16, the UINT is wParam, the BOOL is the HIWORD of lParam, and the HWND is the LOWORD of lParam.

```
LRESULT v_U_B_W_Dispatch(GENERIC& generic,
void (GENERIC::*pmf) (UINT, BOOL, HWND),
WPARAM wParam, LPARAM lParam);
```

v_U_POINT_Dispatch

dispatch.h

Passes a UINT and a reference to a POINT and always returns 0. UINT is wParam and POINT is lParam.

```
LRESULT v_U_POINT_Dispatch(GENERIC& generic,
void (GENERIC::*pmf) (UINT, TPoint &),
WPARAM wParam, LPARAM lParam);
```

v_U_U_Dispatch

dispatch.h

Passes two UINTs and always returns 0. Under WIN 32, the first UINT is wParam and the second UINT is lParam. Under WIN16, the first UINT is wParam and the second UINT is the LOWORD of lParam.

```
LRESULT v_U_U_Dispatch(GENERIC& generic,
void (GENERIC::*pmf) (UINT, UINT), WPARAM wParam,
LPARAM lParam);
```

v_U_U_U_Dispatch**dispatch.h**

Passes three UINTs and always returns 0. The first UINT is wParam, the second UINT is the LOWORD of lParam, and the third UINT is the HIWORD of lParam.

```
LRESULT v_U_U_U_Dispatch(GENERIC& generic,
                        void (GENERIC::*pmf) (UINT, UINT, UINT),
                        WPARAM wParam, LPARAM lParam);
```

v_U_U_W_Dispatch**dispatch.h**

Passes two UINTs and an HWND and always returns 0. In WIN 32, the first UINT is the LOWORD of wParam, the second UINT is the HIWORD of wParam, and the HWND is lParam. In WIN16, the first UINT is wParam, the second UINT is the LOWORD of lParam, and the HWND is the HIWORD of lParam.

```
LRESULT v_U_U_W_Dispatch(GENERIC& generic,
                        void (GENERIC::*pmf) (UINT, UINT, HWND),
                        WPARAM wParam, LPARAM lParam);
```

v_Dispatch**dispatch.h**

Passes no arguments and always returns 0.

```
LRESULT v_v_Dispatch(GENERIC& generic, void (GENERIC::*pmf) (),
                    WPARAM wParam, LPARAM lParam);
```

v_WPARAM_Dispatch**dispatch.h**

Passes WPARAM (ignores LPARAM) as the only parameter and always returns 0.

```
LRESULT v_WPARAM_Dispatch(GENERIC& generic, void (GENERIC::*pmf) (WPARAM),
                        WPARAM wParam, LPARAM lParam);
```

v_WPARAM_LPARAM_Dispatch**dispatch.h**

Does no message cracking (passes wParam and lParam) and always returns 0.

```
LRESULT v_WPARAM_LPARAM_Dispatch(GENERIC& generic,
                                  void (GENERIC::*pfn) (WPARAM, LPARAM),
                                  WPARAM wParam, LPARAM lParam);
```

v_W_W_Dispatch**dispatch.h**

Specifically designed to handle *EvMDIActivate* messages, *v_W_W_Dispatch* passes two HWNDs and always returns 0. Under WIN32, the first HWND is lParam and the second HWND is wParam. Under WIN16, the first HWND is the LOWORD of lParam, and the second HWND is the HIWORD of lParam if lParam is nonzero; otherwise, it is wParam.

```
LRESULT v_W_W_Dispatch(GENERIC& generic, void (GENERIC::*pfn) (HWND, HWND),
                       WPARAM wParam, LPARAM lParam);
```


WIN API encapsulated functions

This chapter includes several tables listing encapsulated Windows API functions that take an HWND as the first argument. The tables are organized in the following manner:

Table	Description	Page
4.1	Inline HWND functions	474
4.2	Windows messages	474
4.3	Window dimensions	474
4.4	Window properties	475
4.5	Window placement	475
4.6	Child window placement	476
4.7	Window painting	476
4.8	Using scrolling and scroll bars	476
4.9	Parent and child windows with IDs	477
4.10	Menus and menu bars	477
4.11	Clipboard placement	477
4.12	Timer operations	478
4.13	Caret and cursor functions	478
4.14	Registering hot keys	478
4.15	Miscellaneous functions	478

Most of the encapsulated functions are implemented as inline functions that pass the `HWindow` member variable as the `HWND` argument. The remaining arguments are passed without changing their prototype.

The other functions are static functions that return information regarding windows. These functions don't actually use `HWindow` as the argument because the `HWND` is either implied or the function returns the `HWND`. An `HWND` that serves as a handle to an `ObjectWindows` window can be converted to a `TWindow*` by using `GetWindowPtr()`.

Note that in the scope of `TWindow` or derived class, all direct calls to the corresponding Windows versions of these functions must be globally scoped, as in `::SendMessage()`.

The functions in Table 4.1 allow a *TWindow* to be used as a *HWND* in Windows API calls.

Table 4.1: Encapsulated inline *HWND* functions

Name	Function declaration
<i>HWND</i>	operator <i>HWND</i> () const
<i>IsWindow</i>	BOOL <i>IsWindow</i> () const

The functions in Table 4.2 handle Window messages.

Table 4.2: Encapsulated Window messages

Name	Function declaration
<i>EnableWindow</i>	BOOL <i>EnableWindow</i> (BOOL enable)
<i>GetCapture</i>	static <i>HWND</i> <i>GetCapture</i> ()
<i>GetFocus</i>	static <i>HWND</i> <i>GetFocus</i> ()
<i>IsWindowEnabled</i>	BOOL <i>IsWindowEnabled</i> () const
<i>PostMessage</i>	BOOL <i>PostMessage</i> (UINT msg, WPARAM wParam = 0, LPARAM lParam = 0) const
<i>ReleaseCapture</i>	static void <i>ReleaseCapture</i> ()
<i>SendDlgItemMessage</i>	LRESULT <i>SendDlgItemMessage</i> (int childId, UINT msg, WPARAM wParam = 0, LPARAM lParam = 0) const
<i>SendMessage</i>	LRESULT <i>SendMessage</i> (UINT msg, WPARAM wParam = 0, LPARAM lParam = 0) const
<i>SetCapture</i>	<i>HWND</i> <i>SetCapture</i> ()
<i>SetFocus</i>	<i>HWND</i> <i>SetFocus</i> ()

Table 4.3 lists functions that adjust window coordinates and sizes.

Table 4.3: Window coordinates and dimensions

Name	Function declaration
<i>AdjustWindowRect</i>	static void <i>AdjustWindowRect</i> (TRect& rect, DWORD style, BOOL menu)
<i>AdjustWindowRectEx</i>	static void <i>AdjustWindowRectEx</i> (TRect& rect, DWORD style, BOOL menu, DWORD exStyle)
<i>ChildWindowFromPoint</i>	<i>HWND</i> <i>ChildWindowFromPoint</i> (const TPoint& point)
<i>ClientToScreen</i>	void <i>ClientToScreen</i> (TPoint& point) const
<i>GetClientRect</i>	TRect <i>GetClientRect</i> ()
<i>GetClientRect</i>	void <i>GetClientRect</i> (TRect& rect)
<i>GetWindowRect</i>	void <i>GetWindowRect</i> (TRect& rect)
<i>MapWindowPoints</i>	void <i>MapWindowPoints</i> (<i>HWND</i> hWndTo, TPoint *points, int count) const
<i>ScreenToClient</i>	void <i>ScreenToClient</i> (TPoint& point) const
<i>WindowFromPoint</i>	static <i>HWND</i> <i>WindowFromPoint</i> (const TPoint& point)

Table 4.4 lists functions that encapsulate window properties and style attributes.

Table 4.4: Window properties

Name	Function declaration
EnumProps	int EnumProps (PROPENUMPROC proc)
GetClassLong	long GetClassLong (int index) const
GetClassName	long GetClassName (char far* className, int maxCount) const
GetClassWord	WORD GetClassWord (int index) const
GetProp	HANDLE GetProp (const char far* str) const
GetProp	HANDLE GetProp (WORD atom) const
GetWindowLong	long GetWindowLong (int index) const
GetWindowWord	WORD GetWindowWord (int index) const
RemoveProp	HANDLE RemoveProp (const char far* str) const
RemoveProp	HANDLE RemoveProp (WORD atom) const
SetClassLong	long SetClassLong (int index, long newLong)
SetClassWord	WORD SetClassWord (int index, WORD newWord)
SetProp	BOOL SetProp (const char far* str, HANDLE data) const
SetProp	BOOL SetProp (WORD atom, HANDLE data) const
SetWindowLong	long SetWindowLong (int index, long newLong)
SetWindowWord	WORD SetWindowWord (int index, WORD newWord)

Table 4.5 lists functions that encapsulate window placement and display properties.

Table 4.5: Window placement

Name	Function declaration
GetWindowPlacement	BOOL GetWindowPlacement (WINDOWPLACEMENT *place) const
GetWindowText	int GetWindowText (char far* str, int maxCount) const
GetWindowTextLength	int GetWindowTextLength () const
IsIconic	BOOL IsIconic () const
IsWindowVisible	BOOL IsWindowVisible () const
IsZoomed	BOOL IsZoomed () const
MoveWindow	void MoveWindow (const TRect& rect, BOOL repaint = FALSE)
MoveWindow	void MoveWindow (int x, int y, int w, int h, BOOL repaint = FALSE)
SetWindowPlacement	BOOL SetWindowPlacement (const WINDOWPLACEMENT *place)
SetWindowText	void SetWindowText (const char far* str)
ShowOwnedPopups	void ShowOwnedPopups (BOOL show)
ShowWindow	BOOL ShowWindow (int cmdShow)

The functions in Table 4.6 control window positions and sibling relationships.

Table 4.6: Window relationships

Name	Function declaration
BringWindowToTop	void BringWindowToTop ()
GetActiveWindow	static HWND GetActiveWindow ()
GetDesktopWindow	static HWND GetDesktopWindow ()
GetLastActivePopup	HWND GetLastActivePopup () const
GetNextWindow	HWND GetNextWindow (UINT dirFlag) const
GetSysModalWindow	static HWND GetSysModalWindow ()
GetTopWindow	HWND GetTopWindow () const
SetActiveWindow	HWND SetActiveWindow ()
SetSysModalWindow	HWND SetSysModalWindow ()
SetWindowPos	void SetWindowPos (HWND hWndInsertAfter, const TRect& rect, UINT flags)
SetWindowPos	void SetWindowPos (HWND hWndInsertAfter, int x, int y, int w, int h, UINT flags)

The encapsulated functions in Table 4.7 control window painting, invalidating, validating, and updating.

Table 4.7: Window painting functions

Name	Function declaration
FlashWindow	BOOL FlashWindow (BOOL invert)
GetUpdateRect	BOOL GetUpdateRect (TRect& rect, BOOL erase = TRUE) const
Invalidate	void Invalidate (BOOL erase = TRUE)
InvalidateRect	void InvalidateRect (const TRect& rect, BOOL erase = TRUE)
InvalidateRgn	void InvalidateRgn (HRGN hRgn, BOOL erase = TRUE)
LockWindowUpdate	BOOL LockWindowUpdate ()
RedrawWindow	BOOL RedrawWindow (TRect *update, HRGN hUpdateRgn, UINT redrawFlags = RDW_INVALIDATE RDW_UPDATENOW RDW_ERASE)
UpdateWindow	void UpdateWindow ()
Validate	void Validate ()
ValidateRect	void ValidateRect (const TRect& rect)
ValidateRgn	void ValidateRgn (HRGN hRgn)

The functions in Table 4.8 control window scrolling and scroll bars.

Table 4.8: Window scrolling functions

Name	Function declaration
GetScrollPos	int GetScrollPos (int bar)
GetScrollRange	void GetScrollRange (int bar, int &minPos, int &maxPos) const
ScrollWindow	void ScrollWindow (int dx, int dy, const TRect *scroll = 0, const TRect *clip = 0)
ScrollWindowEx	void ScrollWindowEx (int dx, int dy, const TRect *scroll = 0, const TRect *clip = 0, HRGN hUpdateRgn = 0, TRect *update = 0, UINT flags = 0)

Table 4.8: Window scrolling functions (continued)

SetScrollPos	int SetScrollPos (int bar, int pos, BOOL redraw = TRUE)
SetScrollRange	void SetScrollRange (int bar, int minPos, int maxPos, BOOL redraw = TRUE)
ShowScrollBar	void ShowScrollBar (int bar, BOOL show = TRUE)

The functions in Table 4.9 control parent and child windows using command IDs.

Table 4.9: Child window ID functions

Name	Function declaration
CheckDlgButton	void CheckDlgButton (int buttonId, UINT check)
CheckRadioButton	void CheckRadioButton (int firstButtonId, int lastButtonId, int checkButtonId)
GetDlgCtrlID	int GetDlgCtrlID () const
GetDlgItem	HWND GetDlgItem (int childId) const
GetDlgItemInt	UINT GetDlgItemInt (int childId, BOOL* translated, BOOL isSigned) const
GetDlgItemText	int GetDlgItemText (int childId, char far* text, int max) const
GetNextDlgGroupItem	HWND GetNextDlgGroupItem (HWND hWndCtrl, BOOL previous = FALSE) const
GetNextDlgTabItem	HWND GetNextDlgTabItem (HWND hWndCtrl, BOOL previous = FALSE) const
GetParent	HWND GetParent () const
IsChild	BOOL IsChild (HWND) const
IsDlgButtonChecked	UINT IsDlgButtonChecked (int buttonId) const
SetDlgItemInt	void SetDlgItemInt (int childId, UINT value, BOOL isSigned = TRUE) const
SetDlgItemText	void SetDlgItemText (int childId, const char far* text) const

The functions in Table 4.10 control menus and menu bar operations.

Table 4.10: Menu and menu bar functions

Name	Function declaration
DrawMenuBar	void DrawMenuBar ()
GetMenu	HMENU GetMenu ()
GetSystemMenu	HMENU GetSystemMenu (BOOL revert = FALSE)
HitTestMenuItem	BOOL HitTestMenuItem (HMENU, UINT idItem, UINT hitTest)
SetMenu	BOOL SetMenu (HMENU)

The function in Table 4.11 controls Clipboard operations.

Table 4.11: Clipboard functions

Name	Function declaration
&OpenClipboard	TClipboard &OpenClipboard ()

The functions in Table 4.12 control timer operations.

Table 4.12: Timer functions

Name	Function declaration
KillTimer	BOOL KillTimer (UINT timerId)
SetTimer	BOOL SetTimer (UINT timerId, UINT timeout, TIMERPROC proc = 0)

The functions in Table 4.13 control caret and cursor operations.

Table 4.13: Caret and cursor functions

Name	Function declaration
CreateCaret	void CreateCaret (HBITMAP)
CreateCaret	void CreateCaret (int shade, int width, int height)
DestroyCaret	static void DestroyCaret ()
GetCaretBlinkTime	static UINT GetCaretBlinkTime ()
GetCaretPos	static void GetCaretPos (TPoint& pos)
GetCursorPos	static void GetCursorPos (TPoint& pos)
HideCaret	void HideCaret ()
SetCaretBlinkTime	static void SetCaretBlinkTime (WORD milliSecs)
SetCaretPos	static void SetCaretPos (const TPoint& pos)
SetCaretPos	static void SetCaretPos (int x, int y)

The functions in Table 4.14 control the operation of hot keys.

Table 4.14: Hot key functions

Name	Function declaration
RegisterHotKey	BOOL RegisterHotKey (int idHotKey, UINT modifiers, UINT virtKey) ¹
UnregisterHotKey	BOOL UnregisterHotKey (int idHotKey) ¹

¹ WIN32 API only

The functions in Table 4.15 control miscellaneous operations such as accessing WinHelp.

Table 4.15: Help and task functions

Name	Function declaration
DragAcceptFiles	void DragAcceptFiles (BOOL accept)
GetWindowTask	HANDLE GetWindowTask () const ¹
GetWindowTask	HTASK GetWindowTask () const ²
MessageBox	int MessageBox (const char far* text, const char far* caption = 0, UINT type = MB_OK)
WinHelp	BOOL WinHelp (const char far* helpFile, UINT command, DWORD data)

¹ WIN32 API only
² WIN16 API only

Inheritance diagrams

This chapter contains the ObjectWindows inheritance diagrams arranged in alphabetical order. Use these diagrams to see the functions that are new or redefined for a particular class. You can tell which virtual functions are inherited by a derived class and which virtual functions are declared in the most derived class. The most derived class is the one enclosed in double lines.

Functions that are defined as virtual in the derived class are underlined. Functions that are declared in a parent class and overridden in a derived class are shaded in the parent class. By definition, these shaded functions are also defined as virtual functions. For example, the virtual function *TGadgetWindow::Paint* overrides the virtual function *TWindow::Paint*.

Application

applicat.h

TModule

AccessResource	GetModuleUsage	LoadResource
AllocResource	GetName	LoadString
CopyCursor	GetParentObject	LowMemory
CopyIcon	GetProcAddress	MakeWindow
Error	InitModule	RestoreMemory
ExecDialog	IsLoaded	SetInstance
FindResource	LoadAccelerators	SetResourceHandler
GetClassInfo	LoadBitmap	SizeOfResource
GetInstance	LoadCursor	ValidWindow
GetInstanceData	LoadIcon	
GetModuleFileName	LoadMenu	

TApplication

BeginModal	GetWinParams	ResumeThrow
BWCCEnabled	IdleAction	Run
CanClose	<u>InitApplication</u>	SetDocManager
Condemn	<u>InitInstance</u>	SetMainWindow
Ct13dEnabled	<u>InitMainWindow</u>	SetMainWinParams
EnableBWCC	MessageLoop	SuspendThrow
EnableCt13dAutosubclass	PostDispatchAction	<u>TermInstance</u>
EndModal	PreProcessMenu	
Find	ProcessAppMsg	
GetDocManager	PumpWaitingMessages	
GetMainWindow	QueryThrow	

TBitmap

TGdiObject

```

GetObject
IsGdiObject
IsOk
RefAdd
RefCount
RefDec
RefFind
RefInc
RefRemove

```

TBitmap

```

BitsPixel
GetBitmapBits
GetBitmapDimension
GetObject
HandleCreate
Height
Planes
SetBitmapBits
SetBitmapDimension
ToClipboard
Width

```

TBitmapGadget

TGadget

CommandEnable	Invalidate	Removed
GetBorders	InvalidateRect	SetBorders
GetBorderStyle	LButtonDown	SetBorderStyle
GetBounds	LButtonUp	SetBounds
GetDesiredSize	MouseEnter	SetEnabled
GetEnabled	MouseLeave	SetMargins
GetId	MouseMove	SetShrinkWrap
GetInnerRect	NextGadget	SetSize
GetMargins	Paint	SysColorChange
GetOuterSizes	PaintBorder	
Inserted	PtIn	

TBitmapGadget

```

GetDesiredSize
Paint
SelectImage
SetBounds
SysColorChange

```

TButton

button.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TButton

BMSetStyle
EvGetDlgCode
GetClassName
SetupWindow

TGadget

CommandEnable	Invalidate	Removed
GetBorders	InvalidateRect	SetBorders
GetBorderStyle	LButtonDown	SetBorderStyle
GetBounds	LButtonUp	SetBounds
GetDesiredSize	MouseEnter	SetEnabled
GetEnabled	MouseLeave	SetMargins
GetId	MouseMove	SetShrinkWrap
GetInnerRect	NextGadget	SetSize
GetMargins	Paint	SysColorChange
GetOuterSizes	PaintBorder	
Inserted	PtIn	

TButtonGadget

Activate	LButtonDown
BeginPressed	LButtonUp
CancelPressed	Paint
CommandEnable	SetButtonState
GetButtonState	SetBounds
GetButtonType	SetNotchCorners
GetDesiredSize	SetShadowStyle
Invalidate	SysColorChange

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupID	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TButton

BMSetStyle
EvGetDlgCode
GetClassName
SetupWindow

TCheckBox

BNClicked	SetCheck
Clicked	SetState
EvGetDlgCode	SetStyle
GetClassName	Toggle
GetCheck	Transfer
GetState	Uncheck

TChooseColorDialog

chooseco.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgnColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetcursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopUp	Previous	WinHelp

TDialog

CloseWindow	EvSetFont
CmCancel	Execute
CmOK	GetClassName
Create	GetDefaultId
Destroy	GetItemHandle
DialogFunction	GetWindowClass
DoCreate	PreProcessMsg
DoExecute	SendDlgItemmsg
EvClose	SetCaption
EvCtlColor	SetDefaultId
EvInitDialog	SetupWindow
EvPaint	

TCommonDialog

CmHelp
CmOkCancel
DoCreate
DoExecute
EvClose
SetupWindow

TChooseColorDialog

DialogFunction
DoExecute
EvSetRGBColor
SetRGBColor

TWindow

AdjustWindowRect	GetSystemMenu	ScrollWindowEx
AdjustWindowRectEx	GetTopWindow	SendDlgItemMessage
BringWindowToTop	GetUpdateRect	SendMessage
CanClose	GetUpdateRgn	SendNotification
CheckDlgButton	GetWindowClass	SetActiveWindow
CheckRadioButton	GetWindowLong	SetBkndColor
ChildBroadcastMessage	GetWindowPlacement	SetCaption
ChildWindowFromPoint	GetWindowPtr	SetCaretBlinkTime
ChildWithId	GetWindowRect	SetCaretPos
CleanupWindow	GetWindowTask	SetClassLong
ClearFlag	GetWindowText	SetClassWord
ClientToScreen	GetWindowTextLeng	SetCursor
CloseWindow	GetWindowTextTitle	SetDlgItem
CmExist	GetWindowWord	SetDlgItemText
Create	HandleMessage	SetFlag
CreateCaret	HideCaret	SetModule
DefaultProcessing	HiliteMenuItem	SetMenu
DefwindowProc	HoldFocusHwnd	SetParent
Destroy	Invalidate	SetProp
DestroyCaret	InvalidateRect	SetScrollPos
DisableAutoCreate	InvalidateRgn	SetScrollRange
DisableTransfer	IsChild	SetSysModalWindow
DispatchScroll	IsDlgButtonChecked	SetTimer
GetChildren	IsFlagSet	SetTransferBuffer
GetClassLong	IsIconic	SetupWindow
GetClassName	IsWindowVisible	SetWindowLong
GetClassWord	IsZoomed	SetWindowPlacement
GetClientRect	KillTimer	SetWindowPos
GetCursorPos	LoadAcceleratorTable	SetWindowText
GetDesktopWindow	LockWindowUpdate	SetWindowWord
GetDlgItem	MapWindowPoints	Show
GetDlgItemInt	MessageBox	ShowCaret
GetDlgItemText	MoveWindow	ShowOwnedPopups
GetFirstChild	Next	ShowScrollBar
GetId	NumChildren	ShowWindow
GetModule	OpenClipboard	SubclassWindowFunction
GetLastActivePopup	Paint	Transfer
GetLastChild	PostMessage	TransferData
GetMenu	PerformCreate	UnregisterHotKey
GetNextDlgGroupItem	PreProcessMsg	UpdateWindow
GetNextDlgTabItem	Previous	Validate
GetNextWindow	ReceiveMessage	ValidateRect
GetObjectPtr	RedrawWindow	ValidateRgn
GetParent	Register	WindowFromPoint
GetProp	RegisterHotKey	WindowProc
GetScrollPos	RemoveProp	WinHelp
GetScrollRange	ScreenToClient	
GetSysModalWindow	ScrollWindow	

TDialog

CloseWindow	EvSetFont
CmCancel	Execute
CmOK	GetClassName
Create	GetDefaultId
Destroy	GetItemHandle
DialogFunction	GetWindowClass
DoCreate	PreProcessMsg
DoExecute	SendDlgItemmsg
EvClose	SetCaption
EvCtlColor	SetDefaultId
EvInitDialog	SetupWindow
EvPaint	

TCommonDialog

```

CmHelp
CmOkCancel
DoCreate
DoExecute
EvClose
SetupWindow

```

TChooseFontDialog

```

CmFontApply
DialogFunction
DoExecute

```

TClipboardViewer

clipview.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TClipboardViewer

EvChangeCBChain
EvDestroy
EvDrawClipboard
SetupWindow

TComboBox

combobox.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
CTearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgnColor
CnExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubClassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TListBox

AddString	GetSel	SetItemData
ClearList	GetSelCount	SetItemHeight
DeleteString	GetSelIndex	SetSel
DirectoryList	GetSelIndexes	SetSelIndex
FindExactString	GetSelString	SetSelIndexes
FindString	GetSelStrings	SetSelItemRange
GetCaretIndex	GetString	SetSelString
GetClassName	GetStringLen	SetSelStrings
GetCount	GetTopIndex	SetTabStops
GetHorizontalExtent	InsertString	SetTopIndex
GetItemData	SetCaretIndex	Transfer
GetItemHeight	SetColumnWidth	
GetItemRect	SetHorizontalExtent	

TComboBox

AddString	GetExtendedUI	SetExtendedUI
Clear	GetItemData	SetItemData
ClearList	GetItemHeight	SetItemHeight
DeleteString	GetSelIndex	SetSelIndex
DirectoryList	GetString	SetSelString
FindString	GetStringLen	SetText
GetClassName	GetText	SetupWindow
GetCount	GetTextLen	ShowList
GetDroppedControlRect	HideList	Transfer
GetDroppedState	InsertString	
GetEditSel	SetEditSel	

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TListBox

AddString	GetSel	SetItemData
ClearList	GetSelCount	SetItemHeight
DeleteString	GetSelIndex	SetSel
DirectoryList	GetSelIndexes	SetSelIndex
FindExactString	GetSelString	SetSelIndexes
FindString	GetSelStrings	SetSelItemRange
GetCaretIndex	GetString	SetSelString
GetClassName	GetStringLen	SetSelStrings
GetCount	GetTopIndex	SetTabStops
GetHorizontalExtent	InsertString	SetTopIndex
GetItemData	SetCaretIndex	Transfer
GetItemHeight	SetColumnWidth	
GetItemRect	SetHorizontalExtent	

TComboBox

AddString	GetExtendedUI	SetExtendedUI
Clear	GetItemData	SetItemData
ClearList	GetItemHeight	SetItemHeight
DeleteString	GetSelIndex	SetSelIndex
DirectoryList	GetString	SetSelString
FindString	GetStringLen	SetText
GetClassName	GetText	SetupWindow
GetCount	GetTextLen	ShowList
GetDroppedControlRect	HideList	Transfer
GetDroppedState	InsertString	
GetEditSel	SetEditSel	

TComboBoxData

AddItemData
AddString
AddStringItem

TCommonDialog

commdial.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgnColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PreProcessMsg	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TDialog

CloseWindow	EvSetFont
CmCancel	Execute
CmOK	GetClassName
Create	GetDefaultId
Destroy	GetItemHandle
DialogFunction	GetWindowClass
DoCreate	PreProcessMsg
DoExecute	SendDlgItemmsg
EvClose	SetCaption
EvCtlColor	SetDefaultId
EvInitDialog	SetupWindow
EvPaint	

TCommonDialog

CmHelp
CmOkCancel
DoCreate
DoExecute
EvClose
SetupWindow

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetcursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetcursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TGadgetWindow

Create	GetInnerRect
EvLButtonDown	GetMargins
EvLButtonUp	IdleActionInsert
EvMouseMove	LayoutSession
EvSize	LayoutUnitsToPixels
EvsysColorChange	NextGadget
FirstGadget	Paint
GadgetChangedSize	PaintGadgets
GadgetFromPoint	PositionGadget
GadgetReleaseCapture	Remove
GadgetSetCapture	SetDirection
GadgetWithId	SetHintCommand
GetDesiredSize	SetHintMode
GetDirection	SetMargins
GetFont	SetShrinkWrap
GetFontHeight	TileGadgets
GetHintMode	

TControlBar

PreProcessMsg
PositionGadget

TControlGadget

controlg.h

TGadget

CommandEnable	Invalidate	Removed
GetBorders	InvalidateRect	SetBorders
GetBorderStyle	LButtonDown	SetBorderStyle
GetBounds	LButtonUp	SetBounds
GetDesiredSize	MouseEnter	SetEnabled
GetEnabled	MouseLeave	SetMargins
GetId	MouseMove	SetShrinkWrap
GetInnerRect	NextGadget	SetSize
GetMargins	Paint	SysColorChange
GetOuterSizes	PaintBorder	
Inserted	PtIn	

TControlGadget

GetDesiredSize
GetInnerRect
Inserted
Invalidate
InvalidateRect
Removed
SetBounds
Update

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgnColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TFrameWindow

AssignMenu	EvQueryDragIcon	PreProcessMsg
EnableKbHandler	EvSetFocus	RestoreMenu
EvCommand	EvSize	SetClientWindow
EvCommandEnable	GetClientWindow	SetIcon
EvEraseBkgn	GetMenuDescr	SetMenuDescr
EvInitMenuPopup	HoldFocusHwnd	SetupWindow
EvPaint	IdleAction	
EvParentNotify	MergeMenu	

TDecoratedFrame

EvCommand	EvSize
EvCommandEnable	Insert
EvEnterIdle	PreProcessMsg
EvMenuSelect	SetupWindow

TDecoratedMDIFrame

decmdifr.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HighlightMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetcursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopUp	Previous	WinHelp

TFrameWindow

AssignMenu	EvQueryDragIcon	PreProcessMsg
EnableKBHandler	EvSetFocus	RestoreMenu
EvCommand	EvSize	SetClientWindow
EvCommandEnable	GetClientWindow	SetIcon
EvEraseBkgnd	GetMenuDescr	SetMenuDescr
EvInitMenuPopup	HoldFocusHwnd	SetupWindow
EvPaint	IdleAction	
EvParentNotify	MergeMenu	

TDecoratedFrame

EvCommand	EvSize
EvCommandEnable	Insert
EvEnterIdle	PreProcessMsg
EvMenuSelect	SetupWindow

TMDIFrame

DefWindowProc
GetClientWindow
SetMenu

TDecoratedMDIFrame

DefWindowProc

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetcursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TDialog

<u>CloseWindow</u>	<u>EvSetFont</u>
<u>CmCancel</u>	<u>Execute</u>
<u>CmOK</u>	<u>GetClassName</u>
<u>Create</u>	<u>GetDefaultId</u>
<u>Destroy</u>	<u>GetItemHandle</u>
<u>DialogFunction</u>	<u>GetWindowClass</u>
<u>DoCreate</u>	<u>PreProcessMsg</u>
<u>DoExecute</u>	<u>SendDlgItemmsg</u>
<u>EvClose</u>	<u>SetCaption</u>
<u>EvCtlColor</u>	<u>SetDefaultId</u>
<u>EvInitDialog</u>	<u>SetupWindow</u>
<u>EvPaint</u>	

TDocManager

docmanag.h

TDocManager

<u>CmFileClose</u>	<u>CreateAnyView</u>	<u>PostDocError</u>
<u>CmFileNew</u>	<u>DeleteTemplate</u>	<u>PostEvent</u>
<u>CmFileOpen</u>	<u>EvCanClose</u>	<u>ReTemplate</u>
<u>CmFileRevert</u>	<u>EvPreProcessMenu</u>	<u>SelectAnySave</u>
<u>CmFileSave</u>	<u>FlushDoc</u>	<u>SelectDocPath</u>
<u>CmFileSaveAs</u>	<u>GetApplication</u>	<u>SelectDocType</u>
<u>CmViewCreate</u>	<u>GetCurrentDoc</u>	<u>SelectViewType</u>
<u>CreateAnyDoc</u>	<u>MatchTemplate</u>	<u>UnReTemplate</u>

TDocTemplate

docmanag.h

TDocTemplate

<u>ClearFlag</u>	<u>GetFileFilter</u>	<u>IsVisible</u>
<u>CreateDoc</u>	<u>GetFlags</u>	<u>SelectSave</u>
<u>CreateView</u>	<u>GetViewName</u>	<u>SetDefaultExt</u>
<u>GetDefaultExt</u>	<u>InitDoc</u>	<u>SetDirectory</u>
<u>GetDescription</u>	<u>InitView</u>	<u>SetDocManager</u>
<u>GetDirectory</u>	<u>IsFlagSet</u>	<u>SetFileFilter</u>
<u>GetDocManager</u>	<u>IsMyKindOfDoc</u>	<u>SetFlag</u>

TDocument

docview.h

TDocument

<u>AttachStream</u>	<u>GetTitle</u>	<u>PropertyFlags</u>
<u>CanClose</u>	<u>HasFocus</u>	<u>PropertyName</u>
<u>Close</u>	<u>InStream</u>	<u>QueryViews</u>
<u>Commit</u>	<u>IsDirty</u>	<u>Revert</u>
<u>DetachStream</u>	<u>IsOpen</u>	<u>RootDocument</u>
<u>FindProperty</u>	<u>NextStream</u>	<u>SetDocManager</u>
<u>GetDocManager</u>	<u>NextView</u>	<u>SetDocPath</u>
<u>GetDocPath</u>	<u>NotifyViews</u>	<u>SetOpenMode</u>
<u>GetOpenMode</u>	<u>Open</u>	<u>SetProperty</u>
<u>GetParentDoc</u>	<u>OutStream</u>	<u>SetTemplate</u>
<u>GetProperty</u>	<u>PostError</u>	<u>SetTitle</u>
<u>GetTemplate</u>	<u>PropertyCount</u>	

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TStatic

Clear
GetClassName
GetText
GetTextLen
SetText
Transfer

TEdit

CanClose	EvGetDlgCode	Insert
CanUndo	EvKeyDown	IsModified
ClearModify	EvKillFocus	LockBuffer
CmEditClear	FormatLines	Paste
CmEditCopy	GetClassName	Scroll
CmEditCut	GetFirstVisibleLine	Search
CmEditDelete	GetHandle	SetHandle
CmEditPaste	GetLine	SetPasswordChar
CmEditUndo	GetLineFromPos	SetReadOnly
Copy	GetLineIndex	SetRect
Cut	GetLineLength	SetRectNP
DeleteLine	GetNumLines	SetSelection
DeleteSelection	GetPasswordChar	SetTabStops
DeleteSubText	GetRect	SetupWindow
EmptyUndoBuffer	GetSelection	SetWordBreakProc
ENErrSpace	GetSubText	Undo
EvChar	GetWordBreakProc	UnlockBuffer

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProperty	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TStatic

Clear
GetClassName
GetText
GetTextLen
SetText
Transfer

TEdit

CanClose	EvGetDlgCode	Insert
CanUndo	EvKeyDown	IsModified
ClearModify	EvKillFocus	LockBuffer
CmEditClear	FormatLines	Paste
CmEditCopy	GetClassName	Scroll
CmEditCut	GetFirstVisibleLine	Search
CmEditDelete	GetHandle	SetHandle
CmEditPaste	GetLine	SetPasswordChar
CmEditUndo	GetLineFromPos	SetReadOnly
Copy	GetLineIndex	SetRect
Cut	GetLineLength	SetRectNP
DeleteLine	GetNumLines	SetSelection
DeleteSelection	GetPasswordChar	SetTabStops
DeleteSubText	GetRect	SetupWindow
EmptyUndoBuffer	GetSelection	SetWordBreakProc
ENErrSpace	GetSubText	Undo
EvChar	GetWordBreakProc	UnlockBuffer

TEditSearch

CmEditFind
CmEditFindNext
CmEditReplace
DoSearch
EvFindMsg
SetupWindow

TEditFile

CanClear	CmFileSaveAs	Save
CanClose	NewFile	SaveAs
CmFileNew	Open	SetFileName
CmFileOpen	Read	SetupWindow
CmFileSave	ReplaceWith	Write

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProperty	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TStatic

Clear
GetClassName
GetText
GetTextLen
SetText
Transfer

TEdit

CanClose	EvGetDlgCode	Insert
CanUndo	EvKeyDown	IsModified
ClearModify	EvKillFocus	LockBuffer
CmEditClear	FormatLines	Paste
CmEditCopy	GetClassName	Scroll
CmEditCut	GetFirstVisibleLine	Search
CmEditDelete	GetHandle	SetHandle
CmEditPaste	GetLine	SetPasswordChar
CmEditUndo	GetLineFromPos	SetReadOnly
Copy	GetLineIndex	SetRect
Cut	GetLineLength	SetRectNP
DeleteLine	GetNumLines	SetSelection
DeleteSelection	GetPasswordChar	SetTabStops
DeleteSubText	GetRect	SetupWindow
EmptyUndoBuffer	GetSelection	SetWordBreakProc
ENErrSpace	GetSubText	Undo
EvChar	GetWordBreakProc	UnlockBuffer

TEditSearch

CmEditFind
CmEditFindNext
CmEditReplace
DoSearch
EvFindMsg
SetupWindow

TEditView

editview.h

TView

FindProperty	IsOK
GetDocument	NotOK
GetNextViewId	PropertyCount
GetProperty	PropertyFlags
GetViewId	PropertyName
GetViewMenu	SetDocTitle
GetViewName	SetProperty
GetWindow	SetViewMenu

TEditSearch

CmEditFind
CmEditFindNext
CmEditReplace
DoSearch
EvFindMsg
SetupWindow

TEditView

CanClose	LoadData	VnDocClosed
Create	PerformCreate	VnIsDirty
EvNCDestroy	SetDocTitle	VnIsWindow
GetViewName	StaticName	VnRevert
GetWindow	VnCommit	

TFileDocument

filedoc.h

TDocument

AttachStream	GetTitle	PropertyFlags
CanClose	HasFocus	PropertyName
Close	InStream	QueryViews
Commit	IsDirty	Revert
DetachStream	IsOpen	RootDocument
FindProperty	NextStream	SetDocManager
GetDocManager	NextView	SetDocPath
GetDocPath	NotifyViews	SetOpenMode
GetOpenMode	Open	SetProperty
GetParentDoc	OutStream	SetTemplate
GetProperty	PostError	SetTitle
GetTemplate	PropertyCount	

TFileDocument

Close	InStream	PropertyFlags
CloseThisFile	IsOpen	PropertyName
Commit	Open	Revert
FindProperty	OpenThisFile	SetProperty
GetProperty	OutStream	

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIcomic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetcursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TDialog

CloseWindow	EvSetFont
CmCancel	Execute
CmOK	GetClassName
Create	GetDefaultId
Destroy	GetItemHandle
DialogFunction	GetWindowClass
DoCreate	PreProcessMsg
DoExecute	SendDlgItemMsg
EvClose	SetCaption
EvCtlColor	SetDefaultId
EvInitDialog	SetupWindow
EvPaint	

TCommonDialog

CmHelp
CmOkCancel
DoCreate
DoExecute
EvClose
SetupWindow

TOpenSaveDialog

CmLbSelChanged
CmOk
DialogFunction
DoExecute
GetFileTitle
GetFileTitleLen
Init
ShareViolation

TFileOpenDialog

DoExecute

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TDialog

CloseWindow	EvSetFont
CmCancel	Execute
CmOK	GetClassName
Create	GetDefaultId
Destroy	GetItemHandle
DialogFunction	GetWindowClass
DoCreate	PreProcessMsg
DoExecute	SendDlgItemmsg
EvClose	SetCaption
EvCtlColor	SetDefaultId
EvInitDialog	SetupWindow
EvPaint	

TCommonDialog

CmHelp
CmOkCancel
DoCreate
DoExecute
EvClose
SetupWindow

TOpenSaveDialog

CmLbSelChanged
CmOk
DialogFunction
DoExecute
GetFileTitle
GetFileTitleLen
Init
ShareViolation

TFileSaveDialog

DoExecute

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TDialog

CloseWindow	EvSetFont
CmCancel	Execute
CmOK	GetClassName
Create	GetDefaultId
Destroy	GetItemHandle
DialogFunction	GetWindowClass
DoCreate	PreProcessMsg
DoExecute	SendDlgItemMsg
EvClose	SetCaption
EvCtlColor	SetDefaultId
EvInitDialog	SetupWindow
EvPaint	

TCommonDialog

CmHelp
CmOkCancel
DoCreate
DoExecute
EvClose
SetupWindow

TFindReplaceDialog

CmFindNext
CmReplace
CmReplaceAll
CmCancel
DoCreate
EvNCDestroy
IsValid
IsValidInput

TFindDialog

DoCreate

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupItem	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TTinyCaption

DoCommand	EnableTinyCaption	GetCaptionRect
DoLButtonUp	EvCommand	GetMinBoxRect
DoMouseMove	EvLButtonUp	GetMaxBoxRect
DoNCActivate	EvMouseMove	GetSysBoxRect
DoNCCalcSize	EVNCActivate	PaintButton
DoNCHitTest	EVNCCalcSize	PaintCaption
DoNCLButtonDown	EVNCHitTest	PaintCloseBox
DoNCPaint	EVNCLButtonDown	PaintMaxBox
DoSysCommand	EVNCPaint	PaintMinBox
DoSysMenu	EvSysCommand	PaintSysBox

TFrameWindow

AssignMenu	EvQueryDragIcon	PreProcessMsg
EnableKBHandler	EvSetFocus	RestoreMenu
EvCommand	EvSize	SetClientWindow
EvCommandEnable	GetClientWindow	SetIcon
EvEraseBkgnd	GetMenuDescr	SetMenuDescr
EvInitMenuPopup	HoldFocusHwnd	SetupWindow
EvPaint	IdleAction	
EvParentNotify	MergeMenu	

TFloatingFrame

SetMargins

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TGadgetWindow

Create	GetInnerRect
EvLButtonDown	GetMargins
EvLButtonUp	IdleActionInsert
EvMouseMove	Insert
EvSize	LayoutSession
EvsysColorChange	LayoutUnitsToPixels
FirstGadget	NextGadget
GadgetChangedSize	Paint
GadgetFromPoint	PaintGadgets
GadgetReleaseCapture	PositionGadget
GadgetSetCapture	Remove
GadgetWithId	SetDirection
GetDesiredSize	SetHintCommand
GetDirection	SetHintMode
GetFont	SetMargins
GetFontHeight	SetShrinkWrap
GetHintMode	TileGadgets

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TGauge

EvEraseBkgnd
Paint
PaintBorder

TLayoutWindow

layoutwi.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusWnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TLayoutWindow

EvSize
GetChildLayoutMetrics
Layout
RemoveChildLayoutMetrics
SetChildLayoutMetrics

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProperty	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HitTestMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TListBox

AddString	GetSel	SetItemData
ClearList	GetSelCount	SetItemHeight
DeleteString	GetSelIndex	SetSel
DirectoryList	GetSelIndexes	SetSelIndex
FindExactString	GetSelString	SetSelIndexes
FindString	GetSelStrings	SetSelItemRange
GetCaretIndex	GetString	SetSelString
GetClassName	GetStringLen	SetSelStrings
GetCount	GetTopIndex	SetTabStops
GetHorizontalExtent	InsertString	SetTopIndex
GetItemData	SetCaretIndex	Transfer
GetItemHeight	SetColumnWidth	
GetItemRect	SetHorizontalExtent	

TListView**listview.h****TView**

FindProperty	IsOK
GetDocument	NotOK
GetNextViewId	PropertyCount
GetProperty	PropertyFlags
GetViewId	PropertyName
GetViewMenu	SetDocTitle
GetViewName	SetProperty
GetWindow	SetViewMenu

TListBox

AddString	GetSel	SetItemData
ClearList	GetSelCount	SetItemHeight
DeleteString	GetSelIndex	SetSel
DirectoryList	GetSelIndexes	SetSelIndex
FindExactString	GetSelString	SetSelIndexes
FindString	GetSelStrings	SetSelItemRange
GetCaretIndex	GetString	SetSelString
GetClassName	GetStringLen	SetSelStrings
GetCount	GetTopIndex	SetTabStops
GetHorizontalExtent	InsertString	SetTopIndex
GetItemData	SetCaretIndex	Transfer
GetItemHeight	SetColumnWidth	
GetItemRect	SetHorizontalExtent	

TListView

CanClose	CmEditUndo	SetExtent
CmEditAdd	CmSelChange	StaticName
CmEditCopy	Create	VnCommit
CmEditClear	EvGetDlgCode	VnDocClosed
CmEditCut	GetViewName	VnIsDirty
CmEditDelete	GetWindow	VnIsWindow
CmEditItem	LoadData	VnRevert
CmEditPaste	SetDocTitle	

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorsPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TMDIClient

ArrangeIcons	CmCloseChildren	EvMdiCreate
CascadeChildren	CmCreateChild	GetActiveMDIChild
CloseChildren	CmTileChildren	GetClassName
CmArrangeIcons	CmTileChildrenHoriz	InitChild
CmCascadeChildren	Create	PreProcessMsg
CmChildActionEnable	CreateChild	TileChildren

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopUp	Previous	WinHelp

TTinyCaption

DoCommand	EnableTinyCaption	GetCaptionRect
DoLButtonUp	EvCommand	GetMinBoxRect
DoMouseMove	EvLButtonUp	GetMaxBoxRect
DoNCActivate	EvMouseMove	GetSysBoxRect
DoNCalcSize	EvNCActivate	PaintButton
DoNCHitTest	EvNCCalcSize	PaintCaption
DoNCLButtonDown	EvNCHitTest	PaintCloseBox
DoNCPaint	EvNCLButtonDown	PaintMaxBox
DoSysCommand	EvNCPaint	PaintMinBox
DoSysMenu	EvSysCommand	PaintSysBox

TFrameWindow

AssignMenu	EvQueryDragIcon	PreProcessMsg
EnableKBHandler	EvSetFocus	RestoreMenu
EvCommand	EvSize	SetClientWindow
EvCommandEnable	GetClientWindow	SetIcon
EvEraseBkgnd	GetMenuDescr	SetMenuDescr
EvInitMenuPopup	HoldFocusHwnd	SetupWindow
EvPaint	IdleAction	
EvParentNotify	MergeMenu	

TMDIFrame

DefWindowProc
GetClientWindow
SetMenu

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TButton

BMSetStyle
EvGetDlgCode
GetClassName
SetupWindow

TCheckBox

BNClicked	SetCheck
Check	SetState
EvGetDlgCode	SetStyle
GetClassName	Toggle
GetCheck	Transfer
GetState	Uncheck

TRadioButton

BNClicked

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TScrollBar

DeltaPos	SbPageUp
GetClassName	SbThumbPosition
GetPosition	SbThumbTrack
GetRange	SbTop
SbBottom	SetPosition
SbLineDown	SetRange
SbLineUp	SetupWindow
SbPageDown	Transfer

TSlider

EvEraseBkgnd	EvSize	PosToPoint
EvGetDlgCode	GetBkColor	SetPosition
EvKeyDown	GetPosition	SetRange
EvKillFocus	GetRange	SetRuler
EvLButtonDb1Clk	HitTest	SetupThumbRgn
EvLButtonDown	NotifyParent	SetupWindow
EvLButtonUp	PaintRuler	SlideThumb
EvMouseMove	PaintSlot	SnapPos
EvPaint	PaintThumb	
EvSetFocus	PointToPos	

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetcursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TGadgetWindow

Create	GetInnerRect
EvLButtonDown	GetMargins
EvLButtonUp	IdleActionInsert
EvMouseMove	Insert
EvSize	LayoutSession
EvsysColorChange	LayoutUnitsToPixels
FirstGadget	NextGadget
GadgetChangedSize	Paint
GadgetFromPoint	PaintGadgets
GadgetReleaseCapture	PositionGadget
GadgetSetCapture	Remove
GadgetWithId	SetDirection
GetDesiredSize	SetHintCommand
GetDirection	SetHintMode
GetFont	SetMargins
GetFontHeight	SetShrinkWrap
GetHintMode	TileGadgets

TMessageBar

GetDesiredSize
GetInnerRect
PaintGadgets
SetHintText
SetText

TStatusBar

Insert
PositionGadget
SetModeIndicator
ToggleModeIndicator
SetSpacing

TTextGadget

textgadg.h

TGadget

CommandEnable	Invalidate	Removed
GetBorders	InvalidateRect	SetBorders
GetBorderStyle	LButtonDown	SetBorderStyle
GetBounds	LButtonUp	SetBounds
GetDesiredSize	MouseEnter	SetEnabled
GetEnabled	MouseLeave	SetMargins
GetId	MouseMove	SetShrinkWrap
GetInnerRect	NextGadget	SetSize
GetMargins	Paint	SysColorChange
GetOuterSizes	PaintBorder	
Inserted	PtIn	

TTextGadget

GetDesiredSize
GetText
Invalidate
Paint
SetText

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgnColor
CmdExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HighlightMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetcursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TGadgetWindow

Create	GetInnerRect
EvLButtonDown	GetMargins
EvLButtonUp	IdleActionInsert
EvMouseMove	Insert
EvSize	LayoutSession
EvsysColorChange	LayoutUnitsToPixels
FirstGadget	NextGadget
GadgetChangedSize	Paint
GadgetFromPoint	PaintGadgets
GadgetReleaseCapture	PositionGadget
GadgetSetCapture	Remove
GadgetWithId	SetDirection
GetDesiredSize	SetHintCommand
GetDirection	SetHintMode
GetFont	SetMargins
GetFontHeight	SetShrinkWrap
GetHintMode	TileGadgets

TToolBox

GetDesiredSize
Insert
TileGadgets

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProperty	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetcursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TView

FindProperty	IsOK
GetDocument	NotOK
GetNextViewId	PropertyCount
GetProperty	PropertyFlags
GetViewId	PropertyName
GetViewMenu	SetDocTitle
GetViewName	SetProperty
GetWindow	SetViewMenu

TWindowView

CanClose
GetViewName
GetWindow
SetDocTitle
StaticName

TWindow

window.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupID	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopUp	Previous	WinHelp

Index

- !=
 - TRegion member function 347
- <<
 - TBitmap related operator 42
 - TClipboard related operator 42, 149, 296
 - TDib related operator 149
- 3-D support 139, 143
- ~TDocManager
 - TDocManager class destructor 153
- A**
- abort dialog box 329, 332
 - canceling 330
 - controls 329, 330
 - creating 328
- AbortDoc
 - TPrintDC member function 308
- Above
 - TEdgeConstraint member function 171
- Absolute
 - TEdgeConstraint member function 171
 - TEdgeOrSizeConstraint member function 172, 173
- abstract data validation 391
- accelerator keys
 - resource IDs 23
- accelerator tables 449
 - handles, returning 284
 - loading into memory 284
- AccelTable
 - TWindowAttr structure 449
- access specifiers 3
- AccessResource
 - TModule member function 282
- Activate
 - TButtonGadget member function 53
- AddItem
 - TVbxControl member function 397
- AddItemData
 - TComboBoxData member function 81
- AddString
 - TComboBox member function 77
 - TComboBoxData member function 81
 - TListBox member function 249
 - TListBoxData member function 255
- AddStringItem
 - TComboBoxData member function 82
 - TListBoxData member function 255
- AdjustWindowRect
 - TWindow member function 415
- AdjustWindowRectEx
 - TWindow member function 415
- alias
 - wfAlias constant 449
- Align
 - TTextGadget member function 382
- allocation
 - edit control error message 180
- AllocResource
 - TModule member function 282
- AngleArc
 - TDC member function 93
- AnimatePalette
 - TPalette member function 295
- animation 55
- AppendMenu
 - TMenu member function 271
- applications
 - facilitating 84
- Arc
 - TDC member function 93
- Area
 - TRect member function 340
- ArrangeIcons
 - TMDIClient member function 263
- arrays
 - bitmap images 43
 - cels 55
- ASCII strings
 - sorting 373
- AssignMenu
 - TFrameWindow member function 204
- AtMouse
 - TGadgetWindow data member 218

- AttachStream
 - TDocument data member 168
- Attr
 - TDialog data member 140
 - TWindow data member 413
- attributes
 - caption bars 383
 - client window 263
 - document properties 28
 - edit controls 178
 - setting 174
 - file 288
 - find-and-replace 198
- auto-creation
 - interface elements, enabling 422
 - interface objects, disabling 421
- automatic scrolling 358
- AutoMode
 - TScroller data member 359
- AutoOrg
 - TScroller data member 359
- AutoScroll
 - TScroller member function 360

B

- BandInfo
 - TPrintDC member function 309
- BANDINFOSTRUCT
 - struct 40
- Banding
 - TPrintout data member 332
- banding flags 332
- banding printouts 331, 332
- beep 180
- BeginDocument
 - TPrintout member function 330
- BeginModal
 - TApplication member function 33
- BeginPath
 - TDC member function 93
- BeginPressed
 - TButtonGadget member function 53
- BeginPrinting
 - TPrintout member function 331
- BeginView
 - TScroller member function 360
- Below
 - TEdgeConstraint member function 171
- BF_XXXX button flag constants 16
- bit count
 - NBits 27
- bit masks 449
- BitBlt
 - TDC member function 94
- Bitmap
 - TCelArray data member 57
- bitmap images
 - displaying 43
- bitmaps
 - deleting 55
 - loading into memory 284
 - referencing 56
 - user interface 55
- Bits
 - TDib member function 145
- BitsPixel
 - TBitmap member function 42
- Bkcolor
 - TSlider member function 371
- BkgndBrush
 - TGadgetWindow data member 218
- Black
 - TColor data member 72
- Blue
 - TColor member function 74
- BMSetStyle
 - TButton member function 49
- BNClicked
 - TCheckBox member function 60
 - TRadioButton member function 337
- Border
 - TTinyCaption data member 384
- Borders
 - TGadget data member 212
- borders
 - status bars, setting 377
- BorderStyle
 - TStatusBar data member 377
- BORLANDC\BIN directory
 - and DLLs 11
- BottomLeft
 - TRect member function 340

- BottomRight
 - TRect member function 340
- Bounds
 - TGadget data member 212
- BreakMessageLoop
 - TApplication member function 37
- BringWindowToTop
 - TWindow member function 415
- broadcasting messages 416
- BS_DEFPUSHBUTTON constant 49
- BS_GROUPBOX style 231
- BS_PUSHBUTTON constant 49
- BS_RADIOBUTTON constant 336
- buffer
 - TInputDialog data member 235
- buffers
 - retrieving handles 175, 178
 - text 174, 176
 - transfer, combo boxes 80, 81
 - transfer data 236, 253, 254, 357, 373, 374, 441, 443
 - constants 391
 - enabling 422
 - pointer to 446
 - size, returning 443
 - transferring data to 391
 - writing to 255
- BufferSize
 - TInputDialog data member 235
- BuffSize
 - TFindReplaceDialog::TData member function 199
- _BUILDDOWLDLL macro 28
- button flag constants 16
- Button key arguments with TVBX controls 406
- button settings 50
- BWCC templates 140
- BWCCEnabled
 - TApplication member function 33
- bytes
 - size of transfer data 373, 374, 443

C

- CalcTerm
 - TPXPictureValidator member function 335
- callback functions 286
- CancelPressed
 - TButtonGadget member function 53
- CanClear
 - TEditFile member function 183
- CanClose
 - TApplication member function 33
 - TDocument member function 164
 - TEdit member function 179
 - TEditFile member function 183
 - TEditView member function 187
 - TListView member function 256
 - TWindow member function 416
 - TWindowView member function 451
- CanUndo
 - TEdit member function 174
- caption bars
 - activating 386
 - close boxes 383, 389
 - creating 384
 - closing 384
 - creating 383
 - default 383
 - dimensions
 - returning 385, 387, 388
 - setting 384
 - minimizing 386, 389
 - painting 386, 388, 389
- CaptionFont
 - TTinyCaption data member 384
- CaptionHeight
 - TTinyCaption data member 384
- captions 39, 84, 267
 - dialog box 236
 - document 410
 - setting 143, 438
 - window 412, 414
- Capture
 - TGadgetWindow data member 218
- CaretRect
 - TSlider member function 371
- carets
 - creating system 417
 - position, returning 176, 178
- carriage returns 175
- CascadeChildren
 - TMDIClient member function 264
- cascading windows 264

- case-insensitive searches *178*
- case-sensitive searches *178*
- catch keyword
 - and exceptions *453*
- CBS_AUTOHSCROLL constant *76*
- CBS_DROPDOWN constant *76*
- CBS_DROPDOWNLIST constant *76*
- CBS_OWNERDRAWFIXED constant *76*
- CBS_OWNERDRAWVARIABLE constant *76*
- CBS_SIMPLE constant *76*
- CBS_SORT constant *76*
- cc
 - TChooseColorDialog data member *61*
- CDTitle
 - TCommonDialog data member *82*
- CelOffset
 - TCelArray member function *56*
- CelRect
 - TCelArray member function *56*
- CelSize
 - TCelArray member function *56*
- cf
 - TChooseFontDialog data member *64*
- ChangeModeToPal
 - TDib member function *147*
- ChangeModeToRGB
 - TDib member function *147*
- characters
 - edit control and *174, 176, 181*
 - end-of-line *175*
 - formatting into buffer *174, 176*
 - invalid
 - checking for *196, 380, 392*
 - numeric values *337*
 - picture formats *334*
 - number scrolled *178*
 - searching for
 - specified *335, 336*
 - user-input *236*
 - valid
 - input fields *197*
 - picture formats *335*
- Check
 - TCheckBox member function *59*
- check boxes
 - status *59*
- CheckComplete
 - TPXPictureValidator member function *335*
- CheckDlgButton
 - TWindow member function *416*
- checking user input *196, 260*
 - data entry *391*
 - input fields *379*
 - numeric values *337*
 - picture strings *333*
- checkmarks *16, 231*
 - radio buttons *336*
- CheckMenuItem
 - TMenu member function *271*
- CheckRadioButton
 - TWindow member function *416*
- child list
 - child windows and *417*
 - deleting objects in *415, 421*
 - interface element and *447*
 - interface object IDs and *427*
 - reading windows into *426*
 - removing objects from *440*
 - test iteration and *423, 424*
 - transfer buffer and *446*
 - window construction and *414*
- child lists
 - interface element and *138, 207*
- child menus, MDI windows *267*
- child windows *416*
 - cascading *264*
 - closing *264*
 - creating *141, 239, 241, 261, 263, 264*
 - decorating *138*
 - first *427*
 - iterator member functions *423, 424*
 - last *427*
 - next *434*
 - number of *434*
 - previous *435*
 - tiling *265*
- ChildBroadcastMessage
 - TWindow member function *416*
- ChildDoc
 - TDocument data member *163*
- ChildMenuPos
 - TMDIFrame data member *267*

- ChildWindowFromPoint
 - TWindow member function 416
- ChildWithId
 - TWindow member function 416
- Chord
 - TDC member function 94
- classes
 - access specifiers 3
 - event handling 18
 - icons 2
 - inline functions 3
 - names
 - client window registration 266
 - edit controls registration 181
 - interface objects registration 446
 - list boxes registration 254
 - modal and modeless dialog box 143
 - static control registration 375
 - scope resolution operator 3
 - TBrush 46
 - TColor 72
 - TDropInfo 169
 - TGdiObject 224
 - TIcon 233
 - TMetaFilePict 278
 - TPalette 293
 - TPaletteEntry 297
 - TPen 298
 - TPoint 301
 - TPointer 304
 - TRect 338
 - TRegion 345
 - TResID 350
 - TRgbQuad 352
 - TRgbTriple 353
 - TSize 363
 - windows
 - input dialog 236
 - Windows registration 435
- ClassExpert
 - command notifications 16, 17, 18
 - document styles 20
- _CLASSTYPE macro 449
- CleanupWindow
 - TWindow member function 416, 446
- Clear
 - TComboBox member function 77
 - TStatic member function 374
- ClearDevice
 - TPrinter member function 328
- ClearDevMode
 - TPrintDialog::TData member function 322
- ClearDevNames
 - TPrintDialog::TData member function 322
- ClearFlag
 - TDocTemplate member function 157
 - TWindow member function 417
- clearing bits 45
- ClearList
 - TComboBox member function 77
 - TListBox member function 249
- ClearModify
 - TEdit member function 175
- client window 206
 - arranging icons 263
 - attributes 263
 - cascading children 264
 - closing children 264
 - moving through 203
 - registration class name 266
 - tiling children 265
- client windows 136
 - handles, returning 283
- ClientAttr
 - TMDIClient data member 263
- ClientSize
 - TLayoutWindow data member 248
- ClientToScreen
 - TWindow member function 417
- ClientWnd
 - TFrameWindow data member 206
- clip
 - TGadget data member 208
- Clipboard
 - activating 69
 - text and 175, 177, 178
 - viewer chain 71
 - windows
 - adding 70, 71
 - identifying 68
 - owner 68
 - removing 71
 - Windows API encapsulation 67

- Clone
 - TXInvalidMainWindow public member function 39
 - TXOwl public member function 454
- Close
 - TDocument member function 164
 - TFileDocument member function 192
 - TMetaFileDC member function 278
- close boxes, creating 383, 384, 389
- CloseBox
 - TTinyCaption data member 384
- CloseChildren
 - TMDIClient member function 264
- CloseClipboard
 - TClipboard member function 67
- CloseFigure
 - TDC member function 95
- CloseThisFile
 - TFileDocument member function 194
- CloseWindow
 - TDialog member function 140
 - TWindow member function 417
- CM_ARRANGEICONS message 265
- CM_CASCADECHILDREN message 265
- CM_CLOSECHILDREN message 265
- CM_CREATECHILD message 265
- CM_EXIT message 417
- CM_FILESAVEAS message 183
- CM_TILECHILDREN message 265
- CM_xxxx edit file constants 17
- CM_xxxx edit replace constants 17
- CM_xxxx MDI constants 18
- CMArrangeIcons
 - TMDIClient member function 265
- CmCancel
 - TDialog member function 140
 - TFindReplaceDialog member function 198
- CMCascadeChildren
 - TMDIClient member function 265
- CMChildActionEnable
 - TMDIClient member function 265
- CMCloseChildren
 - TMDIClient member function 265
- CMCreateChild
 - TMDIClient member function 265
- cmdLine
 - TApplication data member 31
- cmdShow
 - TApplication data member 32
- CmEditAdd
 - TListView member functions 257
- CMEditClear
 - TEdit member function 180
- CmEditClear
 - TListView member functions 258
- CmEditCopy
 - TEdit member function 180
 - TListView member functions 257
- CmEditCut
 - TEdit member function 180
 - TListView member functions 258
- CmEditDelete
 - TEdit member function 180
 - TListView member functions 258
- CmEditFind
 - TEditSearch member function 186
- CmEditFindNext
 - TEditSearch member function 186
- CmEditItem
 - TListView member functions 258
- CmEditPaste
 - TEdit member function 180
 - TListView member functions 258
- CmEditReplace
 - TEditSearch member function 186
- CmEditUndo
 - TEdit member function 180
 - TListView member functions 258
- CmExit
 - TWindow member function 417
- CmFileClose
 - TDocManager member function 153
- CmFileNew
 - TDocManager member function 154
 - TEditFile member function 183
- CmFileOpen
 - TDocManager member function 154
 - TEditFile member function 183
- CmFileRevert
 - TDocManager member function 154
- CmFileSave
 - TDocManager member function 154
 - TEditFile member function 183

- CmFileSaveAs
 - TDocManager member function 154
 - TEditFile member function 183
- CmFindNext
 - TFindReplaceDialog member function 198
- CmFontApply
 - TChooseFontDialog member function 64
- CmHelp
 - TCommonDialog member function 83
- CmLbSelChanged
 - TOpenSaveDialog member function 289
- CmOk
 - TDialog member function 141
 - TOpenSaveDialog member function 289
- CmOkCancel
 - TCommonDialog member function 83
- CmReplace
 - TFindReplaceDialog member function 198
- CmReplaceAll
 - TFindReplaceDialog member function 198
- CmSelChange
 - TListView member functions 258
- CmSetup
 - TPrintDialog data member 320
- CMTileChildren
 - TMDIClient member function 265
- CMTileChildrenHoriz
 - TMDIClient member function 266
- CmViewCreate
 - TDocManager member function 154
- Color
 - TChooseColorDialog::TData data member 63
 - TChooseFontDialog::TData data member 65
- color count
 - NColors 27
- color-matching functions 326
- colors
 - selecting 61
- combo boxes
 - creating 76
 - entries, transferring 80, 81
 - lists
 - showing 80
 - owner-draw 84, 85
 - size 85
 - styles 76
- command-based message constants 16
- CommandEnable
 - called by TGadgetWindow 216
 - TButtonGadget member function 51
 - TGadget data member 210
- CommDlgExtendedError codes 199, 321
- Commit
 - TDocument member function 164
 - TFileDocument member function 192
- CompareItem
 - TControl member function 84
- complement operator
 - used in TBitSet 46
- concatenating strings 255
- Condemn
 - TApplication member function 34
- configuring
 - printers 319, 328
- constants
 - buffer, transfer data 391
 - button flags 16
 - command messages 16
 - event 191
 - exception messages 24
 - resource IDs
 - accelerator keys 23
 - menu commands 23
 - TPrintDialog::TData struct 321
 - transfer functions 391
 - TWindow
 - flags 449
 - validator 455
- constraints
 - defined 241
 - edge 171, 241, 244
 - layout 239, 241
- constructor
 - sample 8, 9
 - TDocTemplateT member function 161
 - TInStream class 236
 - TOutputStream class 292
 - TPreviewPage class 307
 - TPrintPreviewDC class 324
- Constructors
 - TBitmapGadget class 44
- constructors
 - TApplication class 32, 33
 - TBitmap 40

TBitSet class 45
TBrush 47
TButton class 49
TCelArray class 55
TCharSet class 57
TCheckBox class 58
TChooseColorDialog class 61
TChooseFontDialog class 64
TClientDC 67
TClipboard class 70
TClipboardViewer class 71
TColor 73
TComboBox class 76, 77
TComboBoxData class 81
TCommonDialog class 82
TControl class 84
TControlGadget class 88
TCreatedDC 89
TCursor 90
TDC 92, 135
TDecoratedFrame class 136
TDecoratedMDIFrame class 138
TDesktopDC 139
TDialog class 140
TDib 146
TDibDC 151, 152
TDocManager class 153
TDocTemplate class 160
TDocument::List class 169
TDocument class 163
TDropInfo 169
TEdit class 174
TEditFile class 182
TEditSearch class 185
TEditView class 187
TEventInfo 191
TFileDocument class 192
TFileOpenDialog class 195
TFileSaveDialog class 195
TFilterValidator class 196
TFindDialog class 197
TFindReplaceDialog class 198
TFont 202
TFrameWindow class 204
TGdiObject 227
TGroupBox class 231
TIC 233

TIcon 234
TInputDialog class 236
TLayoutMetrics class 242
TLayoutWindow class 247
TListBox class 248, 249
TListBoxData structure 255
TListView class 256
TLookupValidator class 260
TMDIChild class 261
TMDIClient class 263
TMDIFrame class 267
TMemoryDC 269
TMenu class 270, 271
TMenuDescr 275
TMetaFileDC 277
TMetaFilePict 278
TModule class 281
TOpenSaveDialog class 288
TPaintDC 292
TPalette 293
TPaletteEntry 298
TPen 299
TPoint 301
TPointer class 304
TPopupMenu class 305
TPrintDC 308
TPrintDialog class 319
TPrinter class 327
TPrinterAbortDlg class 329
TPrintout class 330
TProcInstance class 333
TPXPictureValidator class 333
TRadioButton class 336
TRangeValidator class 337
TRect 339
TRegion 345
TReplaceDialog class 349
TResID 350
TRgbQuad 352
TRgbTriple 353
TScreenDC 354
TScrollBar class 355
TScroller class 359
TSize 363
TSlider class 367
TStatic class 373, 374
TStatusBar class 376

- TStringLookupValidator class 379
- TSystemMenu class 380
- TTinyCaption class 385
- TValidator class 392
- TVbxControl class 396
- TView class 408
- TWindow class 414
- TWindowDC 450, 451
- TWindowView class 451
- ContainerGroup
 - TMenuDescr enum 276
- Contains
 - TRect member function 340
 - TRegion member function 346
- contents
 - summary 1
- Control
 - TControlGadget data member 88
- control bar button gadgets 86
- control IDs
 - TInputDialog 22
- controls 84
 - drawing 85, 86
 - ID
 - retrieval 142
 - managing 230
- converting
 - horizontal scrollbar range value 361
 - range values to scroll values 26
 - scroll values to range values 26
 - vertical scrollbar range value 362
- Copies
 - TPrintDialog::TData struct data member 321
- Copy
 - TEdit member function 175
- CopyCursor
 - TModule member function 282
- CopyIcon
 - TModule member function 282
- copying exception objects 39
- CountClipboardFormats
 - TClipboard member function 68
- cracking messages 457
- cracking Windows messages 465
- Create
 - TDialog member function 141
 - TEditView member function 187
 - TGadgetWindow member function 219
 - TListView member function 256
 - TMDIClient member function 264
 - TPalette member function 297
 - TWindow member function 417
- CreateAbortWindow
 - TPrinter member function 329
- CreateAnyDoc
 - TDocManager member function 154
- CreateAnyView
 - TDocManager member function 154
- CreateCaret
 - TWindow member function 417
- CreateChild
 - TMDIClient member function 264
- CreateChildren
 - TWindow member function 417
- CreateDialogParam function 141
- CreateDoc
 - TDocTemplate member function 158, 161
- CreateView
 - TDocTemplate member function 158
 - TDocTemplateT member function 161
- creating
 - dialog boxes
 - abort 328, 329, 332
 - print setup 319, 321
 - status bars 376
 - creating DCs using TCreateDC 92
 - creating HDCs 89
- CSize
 - TCelArray data member 57
- CTL3D DLL 34
- CTL3D DLL support 139
- Ctl3dEnabled
 - TApplication member function 34
- CurrentPreviewFont
 - TPrintPreviewDC data member 327
- CursorInstance
 - TWindow data member 445
- CursorResId
 - TWindow data member 445
- cursors
 - default 143
 - loading into memory 285
 - position, inserting text at current 177

- CustColors
 - TChooseColorDialog::TData data member 63
- CustomFilter
 - TOpenSaveDialog::TData struct 290
- Cut
 - TEdit member function 175

D

- Data
 - TChooseColorDialog data member 62
 - TChooseFontDialog data member 64
 - TFindReplaceDialog data member 198
 - TOpenSaveDialog data member 288
 - TPrintDialog data member 320
 - TPrinter data member 328
- data
 - Clipboard, retrieving 68
 - manipulating 67
 - retrieving 283
 - storing
 - document modes 27
 - transfer 338, 393
 - transfer mechanism 84
 - validating entries 391
 - input fields 335, 336, 379, 393, 394
 - numeric values only 337
 - picture strings 333
- data member
 - sample 8, 9
- DC
 - TChooseFontDialog::TData data member 65
 - TPrintout data member 332
- DECLARE_RESPONSE_TABLE macro 18
 - TChooseColorDialog class 62
 - TClipboardViewer class 72
- default error handling 280
- default message processing
 - windows 413
- default printers 319, 327
 - updating 329
- DefaultProc
 - TWindow data member 413
- DefaultProcessing
 - TWindow member function 418
- DefaultProtocol
 - TClipboard data member 67

- DefExt
 - TOpenSaveDialog::TData struct 290
- DEFINE_DOC_TEMPLATE_CLASS macro 18
- DEFINE_RESPONSE_TABLE1 macro
 - TChooseFontDialog class 65
- DEFINE_RESPONSE_TABLE2 macro
 - TDecoratedFrame class 138
- DEFINE_RESPONSE_TABLE macro
 - TDocManager class 157
- DEFINE_RESPONSE_TABLE macros 19
- DefWindowProc
 - TDecoratedMDIFrame member function 138
 - TMDIChild member function 262
 - TMDIFrame member function 268
 - TWindow member function 420
- DeleteItem
 - TControl member function 85
- DeleteLine
 - TEdit member function 175
- DeleteMenu
 - TMenu member function 271
- DeleteSelection
 - TEdit member function 175
- DeleteString
 - TComboBox member function 77
 - TListBox member function 249
- DeleteSubText
 - TEdit member function 175
- DeleteTemplate
 - TDocManager member function 155
- DeltaPos
 - TScrollBar member function 355
- describing menu groups 273
- Destroy
 - TDialog member function 141
 - TDocument::List member function 169
 - TMDIChild member function 261
 - TWindow member function 421
- DestroyCaret
 - TWindow member function 421
- destructor
 - sample 8
 - TFileDocument class 192
 - TProcInstance class 333
- destructors
 - TApplication class 33
 - TBitmapGadget class 44

- TCelArray class 55
- TClipboard class 70
- TComboBoxData class 81
- TControlGadget class 88
- TCreatedDC 89
- TCursor 91
- TDC 93
- TDialoq class 140
- TDib 147
- TDocTemplate class 160
- TDocument::List class 169
- TDocument class 163
- TEditFile class 182
- TEditView class 187
- TFrameWindow class 204
- TGdiObject 228
- TIcon 234
- TLayoutWindow class 247
- TListBoxData structure 255
- TListView class 256
- TMDIChild class 261
- TMDIClient class 263
- TMenu class 271
- TMetaFileDC 278
- TMetaFilePict 279
- TModule class 281
- TPaintDC 293
- TPrinter class 327
- TPrintout class 330
- TScroller class 360
- TSlider class 367
- TStringLookupValidator class 379
- TTinyCaption class 385
- TValidator class 392
- TVbxControl class 397
- TView class 408
- TWindow class 415
- TWindowDC 450
- TWindowView class 451
- DetachStream
 - TDocument data member 168
- DeviceCapabilities
 - TPrintDC associated function 309
- DEVMODE structure
 - locking memory 323
 - pointer to 323
 - unlocking memory 323
- DEVNAMES structure
 - locking memory 323
 - pointer to 323
 - unlocking memory 323
- dialog boxes
 - closing 140, 142
 - creating 61, 63, 141
 - abort 328, 329, 332
 - modeless 349
 - print setup 319, 321
 - CTL3D DLL 139
 - executing 141, 142
 - precautions 283
 - file management 182, 195
 - finding text 197
 - initializing 62, 64, 140, 142, 199
 - input 235
 - interface elements 141
 - items
 - changing 185
 - handles 142
 - sending messages to 143
 - messages, processing 141, 142
 - modal 139, 140
 - modeless 61, 63, 139, 140, 197
 - naming 236
- dialog windows, windows class 236
- DialogBoxParam function 142
- DialogFunction
 - TChooseColorDialog member function 62
 - TChooseFontDialog member function 64
 - TDialoq member function 141
 - TFindReplaceDialog member function 199
 - TOpenSaveDialog member function 289
 - TPrintDialog member function 320
- DIB
 - array of cels 55
- Direction
 - TGadgetWindow data member 218
- direction for tiling 383
- DirectoryList
 - TComboBox member function 77
 - TListBox member function 249
- DirtyFlag
 - TDocument data member 168
 - TListView data member 256

- DirtyLayout
 - TGadgetWindow data member 218
- DisableAutoCreate
 - TWindow member function 421
- DisableItem
 - TBitSet member function 45
- DisableTransfer
 - TWindow member function 421
- disabling CTL3D DLL 34
- Dispatch
 - TEventInfo data member 190
- dispatch function
 - description 465
- dispatch functions
 - HBRUSH_HDC_W_U_Dispatch 466
 - i_LPARAM_Dispatch 466
 - i_U_W_U_Dispatch 466
 - i_WPARAM_Dispatch 467
 - LRESULT_WPARAM_LPARAM_Dispatch 467
 - U_Dispatch 468
 - U_POINT_Dispatch 467
 - U_U_U_U_Dispatch 467
 - U_U_U_W_Dispatch 468
 - U_WPARAM_LPARAM_Dispatch 468
 - v_Dispatch 470
 - v_LPARAM_Dispatch 468
 - v_POINT_Dispatch 468
 - v_POINTER_Dispatch 469
 - v_U_B_W_Dispatch 469
 - v_U_POINT_Dispatch 469
 - v_U_U_Dispatch 469
 - v_U_U_U_Dispatch 470
 - v_U_U_W_Dispatch 470
 - v_W_W_Dispatch 471
 - v_WPARAM_Dispatch 470
 - v_WPARAM_LPARAM_Dispatch 471
- Dispatcher
 - TResponseTableEntry data member 351
- dispatching Window messages 465
- DispatchScroll
 - TWindow member function 446
- display contexts
 - window, painting 434
- displaying
 - current state, program 376, 377
- DLLs
 - building 28
 - error handling 280
 - exporting 29
 - importing 29
 - modules
 - handles, returning 286
 - object-oriented stand-in 280
 - wfAlias and 449
 - dmxxxx manager mode constants 19
 - dnxxxx document message enum 20
 - Doc
 - TStream data member 379
 - TView data member 410
 - DOCINFO
 - struct 319
 - DocInfo
 - TPrintDC data member 319
 - DocList
 - TDocManager data member 153
 - DoCommand
 - TTinyCaption member function 385
 - DoCreate
 - TCommonDialog member function 82
 - TDialog member function 141
 - TFindDialog member function 197
 - TFindReplaceDialog member function 199
 - TReplaceDialog member function 349
 - DocTitleIndex
 - TFrameWindow data member 206
 - document manager
 - creating 152
 - document paths
 - and TFileDocument 193
 - documents
 - closing 20
 - creating 20, 157
 - creating templates 160
 - interfaces 407
 - manager mode constants 19
 - message enum 20
 - naming 410
 - open modes
 - ofxxxx constants 27
 - property attributes 28
 - sharing modes, shxxxx constants 30
 - string ID constants 23
 - templates, creating 18
 - view constants 20

- viewing 407, 451
 - without associated files 162
- DoExecute
 - TChooseColorDialog member function 62
 - TChooseFontDialog member function 64
 - TCommonDialog member function 82
 - TDialog member function 141
 - TFileOpenDialog member function 195
 - TFileSaveDialog member function 196
 - TOpenSaveDialog member function 289
 - TPrintDialog member function 319
- DoKeyDown
 - TKeyboardModeTracker data member 238
- DoLButtonUp
 - TTinyCaption member function 385
- DoMouseMove
 - TTinyCaption member function 385
- DoNCActivate
 - TTinyCaption member function 385
- DoNCCalcSize
 - TTinyCaption member function 385
- DoNCHitTest
 - TTinyCaption member function 386
- DoNCLButtonDown
 - TTinyCaption member function 386
- DoNCPaint
 - TTinyCaption member function 386
- DoSearch
 - TEditSearch member function 186
- DoSetFocus
 - TKeyboardModeTracker data member 238
- DoSysCommand
 - TTinyCaption member function 386
- DoSysMenu
 - TTinyCaption member function 386
- DownHit
 - TTinyCaption data member 384
- DPtoLP
 - TDC member function 95
- Drag
 - TVbxControl member function 397
- DragAcceptFiles
 - TWindow member function 421
- DragFinish
 - TDropInfo member function 170
- DragQueryFile
 - TDropInfo member function 170

- DragQueryFileCount
 - TDropInfo member function 170
- DragQueryFileNameLen
 - TDropInfo member function 170
- DragQueryPoint
 - TDropInfo member function 170
- DrawFocusRect
 - TDC member function 95
- DrawIcon
 - TDC member function 95
- drawing
 - owner-draw controls 85, 86
- drawing gauges 224
- DrawItem
 - TControl member function 85
- DRAWITEM message 85
- DrawMenuBar
 - TWindow member function 422
- DrawText
 - TDC member function 95
- dtxxx constants 20
- dual DC synchronizing functions 324, 327
- dynamic-link libraries (DLLs) 280

E

- edge constraints, creating windows 171, 241, 244
- edit controls 173, 185
 - attributes
 - setting 174, 178
 - closing 179
 - first visible line 175
 - formatting rectangle 177, 178
 - handles 178
 - data 175
 - notification codes 180
 - password 176, 178
 - registration class name 181
 - tab stop positions 179
 - text
 - clearing 180
 - copying 175, 180
 - cutting 175, 180
 - deleting 180
 - lines 175
 - getting 176
 - inserting 177
 - limiting 181

- line index 176
- line length 176
- modified 177
- number of lines 176
- pasting 178, 180
- position 174, 176
- scrolling 178
- selected
 - deleting 175
 - getting 177
 - selecting 179
 - undoing 180
- undoing 174, 175, 179
- word break 179
- wordwrapped lines 175, 177
- EditGroup
 - TMenuDescr enum 276
- editing windows
 - resources and associating with objects 174
- Ellipse
 - TDC member function 97
- EmptyClipboard
 - TClipboard member function 68
- EmptyUndoBuffer
 - TEdit member function 175
- EN_ERRSPACE constant 180
- enable 3-D support 143
- EnableAutoCreate
 - TWindow member function 422
- EnableBWCC
 - and TDialog 140
 - TApplication member function 34
- EnableCtl3d
 - TApplication member function 34
- EnableCtl3dAutosubclass
 - TApplication member function 34
- EnableItem
 - TBitSet member function 45
- EnableKBHandler
 - TFrameWindow member function 204
- EnableMenuItem
 - TMenu member function 271
- EnableTinyCaption
 - TTinyCaption member function 386
- EnableTransfer
 - TWindow member function 422
- encapsulated functions
 - GetParent 428
 - GetProp 428, 440
 - WIN API 473
- encapsulation
 - Windows API Clipboard functions 67
- end-of-line characters 175
- END_RESPONSE_TABLE macro 21
- EndDialog function 141
- EndDoc
 - TPrintDC member function 312
- EndDocument
 - TPrintout member function 331
- EndModal
 - TApplication member function 34
- EndPage
 - TPrintDC member function 312
- EndPath
 - TDC member function 97
- EndPrinting
 - TPrintout member function 331
- EndView
 - TScroller member function 360
- ENErrSpace
 - TEdit member function 180
- Entry
 - TEventInfo data member 191
- enum
 - TMeasurementUnits 269
 - TMenuDescr 276
 - TRelationshipUnits 349
 - TWidthHeight 412
 - TWindowFlag 449
- EnumFontFamilies
 - TDC member function 97
- EnumFonts
 - TDC member function 97
- EnumMetaFile
 - TDC member function 98
- EnumObjects
 - TDC member function 98
- EnumProps
 - TWindow member function 422
- Error
 - TChooseColorDialog::TData data member 63
 - TChooseFontDialog::TData data member 65
 - TFilterValidator member function 196

- TFindReplaceDialog::TData member function 199
- TModule member function 282
- TOpenSaveDialog::TData struct 290
- TPrintDialog::TData struct data member 321
- TPrinter data member 328
- TPXPictureValidator member function 334
- TRangeValidator member function 337
- TStringLookupValidator member function 380
- TValidator member function 392
- error codes
 - TFinReplaceDialog::TData structure 199
 - TPrintDialog class 321
 - TPrinter class
 - returning 328
- error messages
 - displaying 282
 - maximum number of characters 26
- error strings 26
- errors
 - default handling 280
- Escape
 - TPrintDC member function 313
- esComplete 191
- esPartial 191
- EV_CHILD_NOTIFY 22
- EV_CHILD_NOTIFY_ALL_CODES 22
- EV_CHILD_NOTIFY_AND_CODE 22
- EV_CHILD_NOTIFY_AT_CHILD 22
- EV_COMMAND 22
- EV_COMMAND_AND_ID 22
- EV_COMMAND_ENABLE 22
- EV_MESSAGE 22
- EV_OWLDOCUMENT 22
- EV_OWLNOTIFY 22
- EV_OWLVIEW 22
- EV_REGISTERED 22
- EV_VBXEVENTINDEX 404
- EV_VBXEVENTNAME 404
- EV_xxxx macros 21
- EvButtonDown
 - TGadgetWindow member function 219
- EvButtonUp
 - TGadgetWindow member function 219
- EvCanClose
 - TDocManager member function 155
- EvChangeCBChain
 - TClipboardViewer member function 71
- EvChar
 - TEdit member function 180
- EvChildInvalid
 - TWindow member function 422
- EvClose
 - TCommonDialog member function 83
 - TDialog member function 142
- EvClose message 142
- EvCommand
 - TDecoratedFrame member function 137
 - TFrameWindow member function 206
 - TPrinterAbortDlg member function 330
 - TTinyCaption member function 387
 - TWindow member function 422
- EvCommandEnable
 - TDecoratedFrame member function 137
 - TFrameWindow member function 206
 - TWindow member function 422
- EvCtlColor
 - TDialog member function 143
- EvCtlColor message 143
- EvDestroy
 - TClipboardViewer member function 71
- EvDrawClipboard
 - TClipboardViewer member function 71
- event constants 191
- event handling
 - finding events 190, 191
- event-handling functions 457
- event IDs
 - and views 455
- event tables 162
- EvEnterIdle
 - TDecoratedFrame member function 137
- eventhan.h
 - event handlers 457
- events
 - handling 18
 - member functions 21
 - non-client 383
- EvEraseBkgnd
 - TFrameWindow member function 206
- TGauge data member 223
- TSlider member function 368

- EvFindMsg
 - TEditSearch member function 186
- EvGetDlgCode
 - TButton member function 49
 - TCheckBox member function 60
 - TEdit member function 181
 - TListView member functions 258
 - TSlider member function 368
- EvInitDialog
 - TDialog member function 142
- EvInitDialog message 142
- EvInitMenuPopup
 - TFrameWindow member function 206
- EvKeyDown
 - TEdit member function 181
 - TKeyboardModeTracker data member 239
 - TSlider member function 368
- EvKillFocus
 - TEdit member function 181
 - TSlider member function 369
- EvLButtonDblClk
 - TSlider member function 369
- EvLButtonDown
 - TSlider member function 369
- EvLButtonUp
 - TSlider member function 369
 - TTinyCaption member function 387
- EvMDIActivate
 - TMDIChild member function 262
- EvMDICreate
 - TMDIClient member function 266
- EvMenuSelect
 - TDecoratedFrame member function 137
- EvMouseMove
 - TGadgetWindow member function 219
 - TSlider member function 369
 - TTinyCaption member function 387
- EvNCActivate
 - TTinyCaption member function 387
- EvNCCalcSize
 - TTinyCaption member function 387
- EvNCDestroy
 - TEditView member function 188
 - TFindReplaceDialog member function 198
- EvNCHitTest
 - TTinyCaption member function 387
- EvNCLButtonDown
 - TTinyCaption member function 387
- EvPaint
 - and gadgets 220
 - TControl member function 85
 - TDialog member function 142
 - TFrameWindow member function 207
 - TSlider member function 370
 - TTinyCaption member function 388
- EvParentNotify
 - TFrameWindow member function 207
- EvPreProcessMenu
 - TDocManager member function 155
- EvQueryDragIcon
 - TFrameWindow member function 207
- EvSetFocus
 - TFrameWindow member function 207
 - TKeyboardModeTracker data member 239
 - TSlider member function 370
- EvSetFont
 - TDialog member function 142
- EvSetRGBColor
 - TChooseColorDialog member function 62
- EvSize
 - TDecoratedFrame member function 138
 - TFrameWindow member function 207
 - TGadgetWindow member function 219
 - TLayoutWindow member function 248
 - TPreviewPage member function 308
 - TSlider member function 370
- EvSyscolorChange
 - TGadgetWindow member function 220
- EvSysCommand
 - TTinyCaption member function 388
 - TWindow member function 423
- EvVbxDispatch
 - TVbxEventHandler class 407
- EvWindowPosChanging
 - TPreviewPage member function 308
- exception
 - TXINVALIDMAINWINDOW 38
 - TXInvalidMainWindow 38
 - using catch keyword 453
 - using try keyword 453
 - exception bit flags 456
 - exception handler
 - precautions 282

- exception handling
 - TStatus 375
 - TWindow 413
- exception-unsafe code 435
- exceptions
 - error strings 26
 - example in PAINT.CPP 229
 - message constants 24
 - TWindow class 448
 - TXCOMPATIBILITY 281
 - TXCompatibility class 452
 - TXGdi class 229
 - TXInvalidModule 284
 - TXInvalidModule class 287
 - TXInvalidWindow class 39
 - TXOwl class 453
 - TXValidator class 395
- ExcludeClipRect
 - TDC member function 98
- ExcludeUpdateRgn
 - TDC member function 99
- ExecDialog
 - TModule member function 283
- ExecPrintDialog
 - TPrinter member function 329
- Execute
 - TDialog member function 142
- ExStyle
 - TWindowAttr structure 449
- ExtFloodFill
 - TDC member function 99
- ExtTextOut
 - TDC member function 99

F

- file buffers 192
- file error
 - opening a file 194
- file handles 194
- file matching pattern 155
- file names
 - returning
 - expanded 283
- FileData
 - TEditFile data member 182

- FileGroup
 - TMenuDescr enum 276
- FileName
 - TEditFile data member 182
 - TOpenSaveDialog::TData struct 291
- files
 - attributes 288
 - control IDs 22
 - ID constants 24
 - managing 182, 195
 - opening 195, 287
 - document modes 27
 - returning information on 24
 - saving 195, 287
 - sharing modes 30
 - viewing 192
- FillPath
 - TDC member function 100
- FillRect
 - TDC member function 100
- FillRgn
 - TDC member function 100
- Filter
 - TOpenSaveDialog::TData struct 291
- filter for file names 292
- filter validators 196
- FilterIndex
 - TOpenSaveDialog::TData struct 291
- Find
 - TApplication member function 35
 - TEventInfo data member 190
- find-and-replace attributes 198
- FindColor
 - TDib member function 147
- FindExactString
 - TListBox member function 249
- FindIndex
 - TDib member function 147
- FindProperty
 - TDocument member function 164
 - TFileDocument member function 192
 - TView member function 409
- FindResource
 - TModule member function 283
- FindString
 - TComboBox member function 77
 - TListBox member function 250

- FindWhat
 - TFindReplaceDialog::TData member function 200
- firing events
 - child windows 403
- firing events and VBX controls 403
- FirstGadget
 - TGadgetWindow member function 215
- FirstThat
 - TWindow member function 423
- Flags
 - TChooseColorDialog::TData data member 63
 - TChooseFontDialog::TData data member 65
 - TColor member function 74
 - TFindReplaceDialog::TData member function 200
 - TOpenSaveDialog::TData struct 290
 - TPrintDialog::TData struct data member 321
- flags 449
 - interface objects setting 439
- FlashWindow
 - TWindow member function 424
- FlattenPath
 - TDC member function 101
- floating frame
 - and TFrameWindow 204
- FloodFill
 - TDC member function 101
- FlushDoc
 - TDocManager member function 155
- focus
 - shifting 85
- Font
 - TGadgetWindow data member 218
- font
 - creating point size 221
 - for gadget windows 221
- FontHeight
 - TGadgetWindow data member 218
- fonts
 - caption bars 384
 - changing 142
 - selecting 63
 - system
 - layout units and 241
- FontType
 - TChooseFontDialog::TData data member 66

- ForceAllBands
 - TPrintout data member 332
- ForEach
 - TWindow member function 424, 425
- FormatLines
 - TEdit member function 175
- ForwardMessage
 - TWindow member function 425
- fr
 - TFindReplaceDialog data member 198
- Frame
 - TTinyCaption data member 384
- FrameRect
 - TDC member function 101
- FrameRgn
 - TDC member function 101
- FromPage
 - TPrintDialog::TData struct data member 322
- function type
 - TActionFunc 424
 - TActionMemFunc 425
 - TCondFunc 83, 423
 - TCondMemFunc 84, 424
- function types
 - TActionFunc 30
 - TActionMemFunc 30
 - TAnyAnyDispatcher 31
 - TAnyPMF 31
- functions
 - message dispatcher 31
 - transfer 391

G

- gadget border styles 209
- gadget interface objects 208
- gadget margins 209
- gadget placement 301
- gadget settings
 - hint mode 232
 - tiling 383
- GadgetChangedSize
 - TGadgetWindow member function 215
- GadgetFromPoint
 - TGadgetWindow member function 215
- GadgetReleaseCapture
 - TGadgetWindow member function 215

- Gadgets
 - TGadgetWindow data member 218
- gadgets
 - creating separators 362
 - setting attributes 214
 - TGadgetWindow objects 214
- GetDataSetCapture
 - TGadgetWindow member function 215
- gauge controls 222
- GDI classes 224
- generic response table entry 190, 191
- GetActiveChild
 - TMDIClient member function 264
- GetActiveWindow
 - TWindow member function 425
- GetApplication
 - TDocManager member function 155
 - TWindow member function 425
- GetAspectRatioFilter
 - TDC member function 101
- GetAttributeHDC
 - TDC member function 136
 - TPrintPreviewDC data member 327
- GetBitmapBits
 - TBitmap member function 42
- GetBitmapDimension
 - TBitmap member function 42
- GetBits
 - TDib member function 148
- GetBkColor
 - TDC member function 101
 - TSlider member function 370
- GetBkMode
 - TDC member function 101
- GetBorders
 - TGadget data member 210
- GetBorderStyle
 - TGadget data member 210
- GetBounds
 - TGadget data member 210
- GetBoundsRect
 - TDC member function 102
- GetBrushOrg
 - TDC member function 102
- GetCaptionRect
 - TTinyCaption member function 388
- GetCaretBlinkTime
 - TWindow member function 425
- GetCaretIndex
 - TListBox member function 250
- GetCaretPos
 - TWindow member function 425
- GetCharABCWidths
 - TDC member function 102
- GetCharWidth
 - TDC member function 102
- GetCheck
 - TCheckBox member function 59
- GetChildLayoutMetrics
 - TLayoutWindow member function 247
- GetChildren
 - TWindow member function 426
- GetClassInfo
 - TModule member function 283
- GetClassLong
 - TWindow member function 426
- GetClassName
 - TButton member function 50
 - TCheckBox member function 60
 - TComboBox member function 80
 - TDialog member function 143
 - TEdit member function 181
 - TGroupBox member function 232
 - TListBox member function 254
 - TMDIClient member function 266
 - TScrollBar member function 357
 - TStatic member function 375
 - TVbxControl member function 401
 - TWindow member function 446
- GetClassWord
 - TWindow member function 426
- GetClientHandle
 - TModule member function 283
- GetClientRect
 - TWindow member function 426
- GetClientWindow
 - TFrameWindow member function 205
 - TMDIFrame member function 268
- GetClipboardData
 - TClipboard member function 68
- GetClipboardFormatName
 - TClipboard member function 68

- GetClipboardOwner
 - TClipboard member function 68
- GetClipboardViewer
 - TClipboard member function 68
- GetClipBox
 - TDC member function 103
- GetClipRgn
 - TDC member function 103
- GetColor
 - TDib member function 148
- GetColors
 - TDib member function 148
- GetCount
 - TComboBox member function 77
 - TListBox member function 250
- GetCurrentDoc
 - TDocManager member function 155
- GetCurrentObject
 - TDC member function 103
- GetCurrentPosition
 - TDC member function 103
- GetCursorPos
 - TWindow member function 426
- GetDCOrg
 - TDC member function 103
- GetDefaultExt
 - TDocTemplate member function 158
- GetDefaultId
 - TDialog member function 142
- GetDefaultPrinter
 - TPrintDialog member function 319
 - TPrinter member function 329
- GetDescription
 - TDocTemplate member function 158
- GetDesiredSize
 - TBitmapGadget member function 44
 - TButtonGadget member function 54
 - TControlGadget member function 88
 - TGadget data member 210
 - TGadgetWindow member function 220
 - TMessageBar member function 277
 - TTextGadget member function 382
 - TToolBox member function 390
- GetDesktopWindow
 - TWindow member function 426
- GetDeviceCaps
 - TDC member function 104
 - TPrintPreviewDC member function 324
- GetDeviceName
 - TPrintDialog::TData member function 322
- GetDevMode
 - TPrintDialog::TData member function 323
- GetDevNames
 - TPrintDialog::TData member function 323
- GetDialogInfo
 - TPrintout member function 331
- GetDIBits
 - TDC member function 104
- GetDirection
 - TGadgetWindow member function 216
- GetDirectory
 - TDocTemplate member function 158
- GetDlgItem
 - TWindow member function 427
- GetDlgItemInt
 - TWindow member function 427
- GetDlgItemText
 - TWindow member function 427
- GetDocManager
 - TApplication member function 35
 - TDocument member function 164
- GetdocManager
 - TDocTemplate member function 158
- GetDocPath
 - TDocument member function 164
- GetDocument
 - TStream member function 379
 - TView member function 409
- GetDriverName
 - TPrintDialog::TData member function 323
- GetDroppedControlRect
 - TComboBox member function 77
- GetDroppedState
 - TComboBox member function 77
- GetEditSel
 - TComboBox member function 78
- GetEnabled
 - TGadget data member 210
- GetEventIndex
 - TVbxControl member function 397
- GetEventName
 - TVbxControl member function 397
- GetExtendedUI
 - TComboBox member function 78

- GetFileFilter
 - TDocTemplate member function 158
- GetFileName
 - TOpenSaveDialog member function 288
- GetFileNameLen
 - TOpenSaveDialog member function 288
- GetFirstChild
 - TWindow member function 427
- GetFirstVisibleLine
 - TEdit member function 175
- GetFlags
 - TDocTemplate member function 158
- GetFont
 - TGadgetWindow member function 216
- GetFontData
 - TDC member function 104
- GetFontHeight
 - TGadgetWindow member function 216
- GetGlyphOutline
 - TDC member function 110
- GetHandle
 - TEdit member function 175
- GetHCTL
 - TVbxControl member function 397
- GetHDC
 - TDC member function 136
- GetHintMode
 - TGadgetWindow member function 216
- GetHorizontalExtent
 - TListBox member function 250
- GetIconInfo
 - TCursor member function 91
 - TIcon member function 235
- GetId
 - TGadget data member 210
 - TWindow member function 427
- GetIndex
 - TDib member function 148
- GetIndices
 - TDib member function 148
- GetInfo
 - TDib member function 148
- GetInfoHeader
 - TDib member function 148
- GetInnerRect
 - TControlGadget member function 88
 - TGadget data member 213
 - TGadgetWindow member function 220
 - TMessageBar member function 277
- GetInstance
 - TModule member function 283
- GetInstanceData
 - TModule member function 283
- GetItemData
 - TComboBox member function 78
 - TListBox member function 250
- GetItemHandle
 - TDialog member function 142
- GetItemHeight
 - TComboBox member function 78
 - TListBox member function 250
- GetItemRect
 - TListBox member function 250
- GetKerningPairs
 - TDC member function 104
- GetLastActivePopup
 - TWindow member function 427
- GetLastChild
 - TWindow member function 427
- GetLine
 - TEdit member function 176
- GetLineFromPos
 - TEdit member function 176
- GetLineIndex
 - TEdit member function 176
- GetLineLength
 - TEdit member function 176
- GetMainWindow
 - TApplication member function 35
- GetMapMode
 - TDC member function 105
- GetMargins
 - TGadget data member 210
 - TGadgetWindow member function 220
- GetMaxBoxRect
 - TTinyCaption member function 388
- GetMenu
 - TWindow member function 427
- GetMenuCheckMarkDimensions
 - TMenu member function 271
- GetMenuDescr
 - TFrameWindow member function 205
- GetMenuItemCount
 - TMenu member function 271

- GetMenuItemID
 - TMenu member function 272
- GetMenuState
 - TMenu member function 272
- GetMenuString
 - TMenu member function 272
- GetMetaFileBits
 - TMetaFilePict member function 280
- GetMetaFileBitsEx
 - TMetaFilePict member function 280
- GetMinBoxRect
 - TTinyCaption member function 388
- GetModule
 - TWindow member function 427
- GetModuleFileName
 - TModule member function 283
- GetModuleUsage
 - TModule member function 284
- GetName
 - TModule member function 284
- GetNearestColor
 - TDC member function 106
- GetNearestPaletteIndex
 - TPalette member function 295
- GetNextDlgGroupItem
 - TWindow member function 428
- GetNextDlgTabItem
 - TWindow member function 428
- GetNextViewId
 - TView member function 409
- GetNextWindow
 - TWindow member function 428
- GetNumEntries
 - TPalette member function 295
- GetNumEvents
 - TVbxControl member function 397
- GetNumLines
 - TEdit member function 176
- GetNumProps
 - TVbxControl member function 397
- GetObject
 - TBitmap member function 42
 - TBrush member function 48
 - TFont member function 203
 - TGdiObject member functions 225
 - TPalette member function 295
 - TPen member function 300
- GetOpenClipboardWindow
 - TClipboard member function 68
- GetOpenMode
 - TDocument member function 165
- GetOuterSizes
 - TGadget data member 211
- GetOutlineTextMetrics
 - TDC member function 106
- GetOutputName
 - TPrintDialog::TData member function 323
- GetPaletteEntries
 - TPalette member function 295
- GetPaletteEntry
 - TPalette member function 295
- GetParent
 - TWindow member function 428
- GetParentDoc
 - TDocument member function 165
- GetParentObject
 - TModule member function 284
- GetPasswordChar
 - TEdit member function 176
- GetPixel
 - TDC member function 106
- GetPolyFillMode
 - TDC member function 106
- GetPosition
 - TScrollBar member function 355
 - TSlider member function 367
- GetPriorityClipboardFormat
 - TClipboard member function 68
- GetProcAddress
 - TModule member function 284
- GetProp
 - TVbxControl member function 397, 398, 399
 - TWindow member function 428, 429
- Getprop
 - TVbxControl member function 398
- GetProperty
 - TDocument member function 165
 - TFileDocument member function 193
 - TView member function 409
- GetPropIndex
 - TVbxControl member function 399
- GetPropName
 - TVbxControl member function 399

- GetPropType
 - TVbxControl member function 399
- GetRange
 - TGauge data member 222
 - TScrollBar member function 355
 - TSlider member function 367
- GetRect
 - TEdit member function 177
- GetRgnBox
 - TRegion member function 347
- GetROP2()
 - TDC member function 106
- GetScrollPos
 - TWindow member function 429
- GetScrollRange
 - TWindow member function 429
- GetSel
 - TListBox member function 250
- GetSelCount
 - TListBox member function 251
- GetSelection
 - TEdit member function 177
- GetSelIndex
 - TComboBox member function 78
 - TListBox member function 251
- GetSelIndexes
 - TListBox member function 251
- GetSelString
 - TListBox member function 251
 - TListBoxData member function 255
- GetSelStringLength
 - TListBoxData member function 255
- GetSelStrings
 - TListBox member function 251
- GetState
 - TCheckBox member function 59
- GetStock
 - TBrush member functions 48
 - TFont member function 203
 - TPalette member function 295
 - TPen member function 300
- GetStretchBltMode
 - TDC member function 106
- GetString
 - TComboBox member function 78
 - TListBox member function 251
- GetStringLen
 - TComboBox member function 78
 - TListBox member function 251
- GetSubMenu
 - TMenu member function 272
- GetSubText
 - TEdit member function 177
- GetSysBoxRect
 - TTinyCaption member function 388
- GetSysModalWindow
 - TWindow member function 429
- GetSystemMenu
 - TWindow member function 429
- GetSystemPaletteEntries
 - TDC member function 107
- GetSystemPaletteUse
 - TDC member function 107
- GetTabbedTextExtent
 - TDC member function 107
- GetTemplate
 - TDocument member function 165
- GetText
 - TComboBox member function 79
 - TStatic member function 374
 - TTextGadget member function 381
- GetTextAlign
 - TDC member function 107
- GetTextCharacterExtra
 - TDC member function 129
- GetTextColor
 - TDC member function 109
- GetTextExtent
 - TDC member function 109
- GetTextFace
 - TDC member function 109
- GetTextLen
 - TComboBox member function 79
 - TStatic member function 374
- GetTextMetrics
 - TDC member function 110
- GetTitle
 - TDocument member function 165
- GetTopIndex
 - TListBox member function 251
- GetTopWindow
 - TWindow member function 429

- GetUpdateRect
 - TWindow member function 429
- GetUpdateRgn
 - TWindow member function 430
- GetValue
 - TGauge data member 222
- GetViewId
 - TView member function 409
- GetViewMenu
 - TView member function 409
- GetViewName
 - TDocTemplate member function 158
 - TDocTemplateT member function 162
 - TEditView member function 187
 - TListView member function 256
 - TView member function 409
 - TWindowView member function 452
- GetViewportExt
 - TDC member function 110
- GetViewportOrg
 - TDC member function 111
- GetWindow
 - TEditView member function 187
 - TListView member function 257
 - TView member function 409
 - TWindowView member function 452
- GetWindowClass
 - TDialog member function 143
 - TWindow member function 446
- GetWindowExt
 - TDC member function 111
- GetWindowLong
 - TWindow member function 430
- GetWindowOrg
 - TDC member function 111
- GetWindowPlacement
 - TWindow member function 430
- GetWindowPtr
 - TWindow member function 428
- GetWindowRect
 - TWindow member function 430
- GetWindowTask
 - TWindow member function 430
- GetWindowText
 - TWindow member function 430
- GetWindowTextLength
 - TWindow member function 431
- GetWindowTextTitle
 - TWindow member function 431
- GetWindowWord
 - TWindow member function 431
- GetWinMainParams
 - TApplication member function 35
- GetWordBreakProc
 - TEdit member function 177
- Graphics Device Interface classes 224
- Gray
 - TColor data member 72
- GrayString
 - TDC member function 111
- Green
 - TColor member function 74
- Group
 - TCheckBox data member 58
 - TPXPictureValidator member function 335
- group boxes
 - creating 230
 - notification 231
 - radio buttons and 336
 - selection
 - changing 231
- Groupcount[NumGroups]
 - TMenuDescr data member 275

H

- H
 - TDib member function 145
 - TWindowAttr structure 450
- hAccel
 - TWindow data member 445
- HAccTable
 - TApplication data member 32
- Handle
 - TDC data member 134
 - TGdiObject data member 227
 - TMenu data member 273
- HandleCreate
 - TBitmap member function 43
- HandleGlobalException 283
- HandleMessage
 - TWindow member function 431
- handles
 - dialog boxes 142

- retrieving *175, 178, 428*
 - Clipboard-viewer *68*
- returning
 - accelerator tables *284*
 - client windows *283*
 - DLLs *286*
 - parent windows *284*
 - resources *285*
 - Windows applications *286*
- handling document views *162*
- handling exceptions
 - using `ReceiveMessage` *435*
- handling input focus *204*
- Has
 - `TBitSet` member function *45*
- HasFocus
 - `TDocument` member function *165*
- HasHScrollBar
 - `TScroller` data member *359*
- HasOption
 - `TValidator` member function *392*
- HasPage
 - `TPrintout` member function *331*
- HasVScrollBar
 - `TScroller` data member *359*
- HBRUSH()
 - `TBrush` member function *48*
- HCursor
 - `TWindow` data member *445*
- HCURSOR[]
 - `TCursor` member function *91*
- header files
 - described *11*
 - summary *11*
- Height
 - `TBitmap` member function *42*
 - `TDib` member function *148*
 - `TLayoutMetrics` data member *242*
 - `TRect` member function *340*
- HelpGroup
 - `TMenuDescr` enum *276*
- HFILE_ERROR *192*
- HFONT()
 - `TFont` member function *203*
- HICON()
 - `TIcon` member function *235*
- HideCaret
 - `TWindow` member function *431*
- HideList
 - `TComboBox` member function *79*
- Highlightline
 - `TMessageBar` data member *277*
- HighValue
 - `TScrollBarData` data member *358*
- HiliteMenuItem
 - `TWindow` member function *432*
- HInstance
 - `TModule` member function *287*
- hint modes *232*
- hint text
 - and `TMessageBar` *276*
- HintMode
 - `TGadgetWindow` data member *218*
- HitTest
 - `THSlider` member function *232*
 - `TSlider` member function *370*
 - `TVSlider` member function *411*
- HMENU
 - `TMenu` member function *272*
- HoldFocusHwnd
 - `TFrameWindow` member function *205*
 - `TWindow` member function *432*
- horizontal scroll bars *354*
- horizontal scrollbar range value
 - converting *361*
- horizontal sliders *232*
- HPALETTE()
 - `TPalette` member function *296*
- HPEN()
 - `TPen` member function *300*
- HPrevInstance
 - `TApplication` data member *32*
- HRGN()
 - `TRegion` member function *348*
- HScroll
 - `TScroller` member function *360*
- HWindow
 - `TWindow` data member *413*
- HWNDNext
 - `TClipboardViewer` data member *72*
- HWNDRestoreFocus
 - `TFrameWindow` data member *206*

I

icons

- arranging 263
- described 2
- loading into memory 285

Id

- TEventInfo data member 191
- TGadget data member 212
- TResponseTableEntry data member 351
- TWindowAttr structure 449

ID_FIRSTMDICHILD constant 263

ID_MDICLIENT constant 263

ID_XXXX file constants 22

ID_XXXX printer constants 23

IDA_XXXX accelerator ID constants 23

IDCANCEL constant 140, 141

IdleAction

- TApplication member function 38
- TFrameWindow member function 205
- TGadgetWindow member function 216

IDM_XXXX menu ID constants 23

IDOK constant 141

IDs

- interface objects 427
- retrieving default 142

IDS_GDIFAILURE

- TXGdi constructor 230

IDS_INVALIDMAINWINDOW

- TXInvalidMainWindow constructor 39

IDS_INVALIDWINDOW

- TXWindow constructor 448

IDS_XXXX document string ID constants 23

IDS_XXXX edit file ID constants 24

IDS_XXXX exception messages 24

IDS_XXXX listview string ID constants 25

IDS_XXXX printer string ID constants 25

IDS_XXXX validator ID constants 25

IDW_MDICLIENT constant 26

IDW_MDIFIRSTCHILD constant 26

Index

- TColor member function 75

index

- list position 249, 250, 251, 252, 253

indicators, status bar 376, 377

- borders 377
- spacing 377, 378

InflatedBy

- TRect member function 341

Info

- TDib member function 145

InfoFromHandle

- TDib member function 151

inheritance diagram

- description 7

TApplication 479

TBitmap 480

TBitmapGadget 480

TButton 481

TButtonGadget 482

TCheckBox 483

TChooseColorDialog 484

TChooseFontDialog 486

TClipboardViewer 487

TComboBox 488

TComboBoxData 490

TCommonDialog 492

TControl 493

TControlBar 494

TControlGadget 495

TDecoratedFrame 496

TDecoratedMDIFrame 497

TDialog 499

TDocManager 500

TDocTemplate 500

TDocument 500

TEdit 501

TEditFile 503

TEditSearch 505

TEditView 506

TFileDocument 507

TFileOpenDialog 508

TFileSaveDialog 510

TFindDialog 512

TFloatingFrame 514

TGadgetWindow 516

TGauge 518

TLayoutWindow 519

TListBox 520

TListView 521

TMDIClient 522

TMDIFrame 523

TRadioButton 525

TSlider 527

- TStatusBar 529
- TTextGadget 530
- TToolBox 531
- TWindow 534
- TWindowView 533
- inherited member functions 8, 9
- Init
 - TDC member function 136
 - TFindReplaceDialog member function 199
 - TOpenSaveDialog member function 289
- InitApplication
 - TApplication member function 38
- InitChild
 - TMDIClient member function 264
- InitDoc
 - TDocTemplate member function 160
- InitialDir
 - TOpenSaveDialog::TData struct 291
- initialization
 - dialog boxes 140, 142
 - instance 31
- InitInstance
 - TApplication member function 38
- InitMainWindow
 - TApplication member function 39
- InitModule
 - TModule member function 284
- InitView
 - TDocTemplate member function 160
- inline functions 3
- inline StartPage
 - TPrintDC member function 318
- input dialog windows 235
- input fields 196
 - checking for invalid entries 196, 260
 - data entry 379, 393, 394
 - picture formats 335, 336
 - comma separators 336
 - defining character sets 197
- input focus 204
- Insert
 - TDecoratedFrame member function 136
 - TDocument::List member function 169
 - TEdit member function 177
 - TGadgetWindow member function 216
 - TStatusBar member function 376
 - TToolBox member function 390
- Inserted
 - TControlGadget member function 88
 - TGadget member function 213
- InsertMenu
 - TMenu member function 272
- InsertString
 - TComboBox member function 79
 - TListBox member function 252
- instance initialization 31
- InStream
 - TDocument member function 165
 - TFileDocument member function 193
- integers
 - range
 - testing for 337
- interface elements
 - auto-creation, disabling 421
 - auto-creation, enabling 422
 - controls 84
 - destroying 421
 - dialog boxes 141
- interface objects
 - child windows 416
 - closing
 - conditional 416
 - data, transferring 443
 - flags, setting 439
 - group boxes 230
 - IDs 427
 - list boxes 248
 - registration class name 446
 - scrolling 446
 - setting up 138, 207, 447
 - showing 442
 - static text 373
 - status 375, 413
 - transfer mechanism
 - disabling 421
 - enabling 422
 - window objects 446
- IntersectClipRect
 - TDC member function 112
- invalid characters
 - checking for 196, 380, 392
 - numeric values 337
 - picture formats 334

- Invalidate
 - TButtonGadget member function *54*
 - TControlGadget member function *88*
 - TGadget data member *213*
 - TTextGadget member function *382*
 - TWindow member function *432*
- InvalidateRect
 - TControlGadget member function *88*
 - TGadget data member *213*
 - TWindow member function *432*
- InvalidateRgn
 - TWindow member function *432*
- invalidating buttons *54*
- InvertRect
 - TDC member function *112*
- InvertRgn
 - TDC member function *112*
- IsArrayProp
 - TVbxControl member function *399*
- IsAutoMode
 - TScroller member function *360*
- IsChild
 - TWindow member function *432*
- IsClipboardFormatAvailable
 - TClipboard member function *69*
- IsCore
 - TDib member function *145*
- IsCurrentDefPB
 - TButton data member *49*
- IsDefPB
 - TButton data member *49*
- IsDirty
 - TDocument member function *165*
- IsDlgButtonChecked
 - TWindow member function *432*
- IsEmpty
 - TBitSet member function *45*
 - TRect member function *341*
- IsFlagSet
 - TWindow member function *432*
- IsFlagSet TDocTemplate member function *158*
- IsGDIObject
 - TGdiObject member functions *226*
- IsHorizontal
 - TGauge data member *223*
- IsIconic
 - TWindow member function *433*
- IsLoaded
 - TModule member function *284*
- IsModal
 - TDialog data member *140*
- IsModified
 - TEdit member function *177*
- IsMyKindOfDoc
 - TDocTemplate member function *159*
 - TDocTemplateT member function *161*
- IsMyKindOfView
 - TDocTemplateT member function *161*
- IsNull
 - TRect member function *341*
- IsOK
 - TDib member function *148*
 - TGdiObject member function *226*
 - TMenu member function *272*
 - TMetaFilePict member function *280*
 - TView member function *409*
- IsOpen
 - TClipboard data member *70*
 - TDocument member function *165*
 - TFileDocument member function *193*
- IsPM
 - TDib member function *149*
- isPressed
 - TTinyCaption data member *384*
- IsString
 - TResID member function *350*
- IsValid
 - TFilterValidator member function *196*
 - TLookupValidator member function *260*
 - TPXPictureValidator member function *334*
 - TRangeValidator member function *337*
 - TValidator member function *393*
- IsValidInput
 - TFilterValidator member function *196*
 - TPXPictureValidator member function *334*
 - TValidator member function *393*
- IsVisible
 - TDocTemplate member function *159*
- IsVisibleRect
 - TScroller member function *360*
- IsWindowVisible
 - TWindow member function *433*
- IsZoomed
 - TWindow member function *433*

- ItemDatas
 - TComboBoxData data member 81
 - TListBoxData data member 254
- Iteration
 - TPXPictureValidator member function 335
- iterator member functions
 - child windows 423

K

- keyboard mode
 - tracking 237
- keyboard navigation
 - TMDI frame 267
- KeyboardHandling
 - TFrameWindow data member 204
- KillTimer
 - TWindow member function 433

L

- Layout
 - TLayoutWindow member function 247
- layout constraints, creating windows 239, 241
- layout units
 - font size 241
 - screen resolution 241
- LayoutSession
 - TGadgetWindow member function 216
- LayoutUnitsToPixels
 - TGadgetWindow member function 220
- LBS_NOTIFY constant 249
- LBS_SORT constant 249
- LBS_SORT style 249
- LButtonDown
 - TButtonGadget member function 54
 - TGadget data member 213
- LButtonUp
 - TButtonGadget member function 54
 - TGadget data member 213
- LedSpacing
 - TGauge data member 223
- LedThick
 - Gauge data member 223
- LeftOf
 - TEdgeConstraint member function 171

- libraries
 - dynamic link
 - summary 11
 - summary 10
 - version number, returning 30
- LineDDA
 - TDC member function 113
- linefeeds 175
- LineMagnitude
 - TScrollBar data member 354
- LineTo
 - TDC member function 113
- list boxes 85, 254
 - clearing 249
 - creating 248
 - entries
 - adding 249
 - deleting 249
 - getting 251
 - initializing 255
 - inserting 252
 - length of 251
 - number of 250, 251
 - selecting 251, 252, 253
 - transferring 253
 - registration class name 254
 - transfer structures 254
- list view constants 25
- LmParent constant 26
- LoadAccelerators
 - TModule member function 284
- LoadAcceleratorTable
 - TWindow member function 446
- LoadBitmap
 - TModule member function 284
- LoadCursor
 - TModule member function 285
- LoadData
 - TEditView member function 188
 - TListView member functions 258
- LoadFile
 - TDib member function 151
- LoadIcon
 - TModule member function 285
- LoadMenu
 - TModule member function 285

- LoadResource
 - TDib member function 151
 - TModule member function 285
- LoadString
 - TModule member function 286
- Lock
 - TPrintDialog::TData member function 323
- LockBuffer
 - TEdit member function 177
- LockWindowUpdate
 - TWindow member function 433
- LogFont
 - TChooseFontDialog::TData data member 66
- Lookup
 - TLookupValidator member function 260
 - TStringLookupValidator member function 380
- lookup validators 260
 - string 379
- LowMemory
 - TModule member function 286
- LowValue
 - TScrollBarData data member 358
- LParam
 - control message parameter 143
- lpCmdLine
 - TModule data member 280
- LPtoDP
 - TDC member function 113
- LPtoSDP
 - TPrintPreviewDC member function 324
- LtBlue
 - TColor data member 72
- LtCyan
 - TColor data member 72
- LtGray
 - TColor data member 72
- LtGreen
 - TColor data member 73
- LtMagenta
 - TColor data member 73
- LtRed
 - TColor data member 73
- LtYellow
 - TColor data member 73

M

- macros
 - DEFINE_RESPONSE_TABLE 19
 - event handling 18, 21
 - TButton class 50
 - TChooseColorDialog class 62
 - TChooseFontDialog class 65
 - TClipboardViewer class 72
 - TDecoratedFrame class 138
 - TDocManager class 157
 - TFloatingFrame class 202
 - TKeyboardModeTracker class 239
 - TTinyCaption class 389
- Magnitude
 - TSize member function 364
- main window
 - closing 34, 417
 - creating 31, 38
 - MDI frame 267
 - naming 39
 - nCmdShow display 32
 - status 281
- maintaining a view's event table 162
- MainWindow
 - variable 39
- MakeWindow
 - TModule member function 286
- MapColor
 - TDib member function 149
- MapIndex
 - TDib member function 149
- MapStatusCodeToString
 - TXCompatibility public member function 453
- MapWindowPoints
 - TWindow member function 433
- Margin
 - TGauge data member 223
- Margin constant 241
- Margins
 - TGadget data member 212
 - TGadgetWindow data member 218
- margins
 - for gadgets 217
- MaskBlt
 - TDC member function 114
- MatchTemplate
 - TDocManager member function 155

- matrix
 - toolbox arrangement *389*
- Max
 - TGauge data member *223*
 - TRangeValidator data member *338*
 - TSlider member function *371*
- MAX_RSRC_ERROR_STRING constant *26*
- maximum values
 - checking for *338*
- MaxPage
 - TPrintDialog::TData struct data member *322*
- MaxWidth
 - TListView data member *257*
- MB_Xxxx constants
 - and document manager *156*
 - and Tdocument *166*
- MDIFILE.CPP
 - illustrating exceptions *453*
- MeasureItem
 - TControl member function *85*
- measurement units, windows *269*
- member functions *8, 9*
 - defining *83, 84*
 - edge constraints
 - determining *173*
 - event handling *21*
 - obsolete *142*
 - pointers
 - generic *31*
- memory
 - freeing *182*
 - managing *280*
- Menu
 - TWindowAttr structure *450*
- menu descriptor
 - deleting *204*
- menu resource ID *450*
- menubar
 - number of items *275*
- MenuItemId
 - TDecoratedFrame data member *137*
- MENUITEMTEMPLATE structures *285*
- menus
 - creating *305, 380*
 - ID constants *23*
 - loading into memory *285*
 - managing *270*
- MDI windows *267*
 - system *380*
- MergeMenu
 - and TMenuDescr *273*
 - TFrameWindow member function *205*
- merging menus *273*
- message bar
 - hint text *276*
- message bar implementation *276*
- message boxes
 - errors *282*
- message crackers *457*
- message dispatcher *31*
- message-handling functions *457*
- MessageBox
 - TWindow member function *433*
- MessageLoop
 - TApplication member function *35*
- MessageLoopResult
 - TApplication member function *37*
- messages *375*
 - exception constants *24*
 - preprocessing *137, 142*
 - processing *141, 413, 418*
 - incoming *444*
 - response *207*
 - sending
 - to dialog boxes *143*
- Method
 - TVbxControl member function *399*
- metrics
 - layout *26*
- Min
 - TGauge data member *223*
 - TRangeValidator data member *338*
 - TSlider member function *371*
- minimum values
 - checking for *338*
- MinPage
 - TPrintDialog::TData struct data member *322*
- modal dialog boxes *139, 140*
- Mode
 - TDib member function *145*
- mode indicators *376, 377*
 - borders *377*
 - spacing *377, 378*

- ModeIndicators
 - TStatusBar data member 377
- modeless dialog boxes
 - creating 349
- Modes
 - TKeyboardModeTracker data member 237
- ModifyMenu
 - TMenu member function 272
- ModifyWorldTransform
 - TDC member function 115
- Module
 - TModule data member 281
- modules 31
 - DLL stand-in 280
 - instance handles 286
 - names 287
- MouseEnter
 - TGadget member function 213
- MouseLeave
 - TGadget member function 213
- MouseMove
 - TButtonGadget member function 54
 - TGadget member function 214
- MouseOffset
 - TSlider member function 371
- Move
 - TVbxControl member function 399
- MoveTo
 - TDC member function 115
- MoveWindow
 - TWindow member function 434
- Msg
 - TEventInfo data member 190
 - TResponseTableEntry data member 351
 - TXGdi member function 230
 - TXWindow member function 448
- multiple document interface
 - child menu 267
 - child windows 261
 - cascading 265
 - closing 265
 - creating 264, 265
 - tiling 265
 - client windows 263
 - freeing 263
 - icons, arranging 265
 - main window 267

- MyEdge
 - TLayoutConstraint data member 240

N

- Name
 - TDialogAttr data member 144
 - TModule member function 287
- named streams
 - and TFileDocument 193
- names
 - modules 287
- NBits 27
- NCels
 - TCelArray data member 57
- nCmdShow
 - TApplication data member 32
- NColors 27
- NewFile
 - TEditFile member function 183
- NewStringList
 - TStringLookupValidator member function 380
- Next
 - TWindow member function 434
- NextBand
 - TPrintDC member function 317
- NextGadget
 - TGadget data member 211
 - TGadgetWindow member function 217
- NextStream
 - TDocument member function 166
 - TStream data member 379
- NextView
 - TDocument member function 166
- non-client events 383
- Normalize
 - TRect member function 341
- Normalized
 - TRect member function 341
- NotchCorners
 - TButtonGadget data member 52
- notification codes 180
- NotifyCodeTResponseTableEntry data member 351
- NotifyParent
 - TGroupBox data member 231
 - TSlider member function 232
 - TSlider member function 370

- TVSlider member function 411
- NotifyViews
 - TDocument member function 166
- NotOK
 - TView data member 410
- NumCels
 - TCelArray member function 56
- NumChars
 - TTextGadget member function 382
- NumChildren
 - TWindow member function 434
- NumClrs
 - TDib member function 146
- numColors
 - TDib member function 149
- NumColumns
 - TToolBox data member 390
- numeric values
 - checking 337, 393
 - ranges
 - testing for 337
 - setting
 - maximum/minimum 338
- NumGadgets
 - TGadgetWindow data member 218
- NumGroups
 - TMenuDescr enum 276
- NumModeIndicators
 - TStatusBar data member 377
- NumRows
 - TToolBox data member 390
- numScans
 - TDib member function 149

O

- OBJ_REF_ADD
 - TGdiObject macro 228
- OBJ_REF_COUNT
 - TGdiObject macro 228
- OBJ_REF_DEC
 - TGdiObject macro 229
- OBJ_REF_INC
 - TGdiObject macro 229
- OBJ_REF_REMOVE
 - TGdiObject macro 229
- Object
 - TEventInfo data member 191

- ObjectGroup
 - TMenuDescr enum 276
- objects
 - windows 446
- ObjectWindows inheritance diagrams 479
- ODADrawEntire
 - TControl member function 85
- ODAFocus
 - TControl member function 85
- ODASelect
 - TControl member function 86
- Offs
 - TCelArray data member 57
- Offset
 - TCelArray member function 56
 - TPoint member function 302
 - TRect member function 342
- OffsetBy
 - TPoint member function 302
 - TRect member function 342
- OffsetClipRgn
 - TDC member function 116
- OffsetViewportOrg
 - TDC member function 116
 - TPrintPreviewDC member function 324
- OffsetWindowOrg
 - TDC member function 116
- ofn
 - TOpenSaveDialog data member 288
- ofxxx document open enum (constants) 27
- OLE programs
 - constants
 - compatible 27
- Open
 - TDocument member function 166
 - TEditFile member function 183
 - TFileDocument member function 193
- OpenClipboard
 - TClipboard member function 69
 - TWindow member function 434
- OPENFILENAME structure 195
- opening a file
 - file error 194
- OpenMode
 - TStream data member 378
- OpenThisFile
 - TFileDocument member function 194

- operator
 - TStatus data member 375
- operator&
 - TRect member function 342
- operator+
 - TPoint member function 302
 - TRect member function 342
 - TSize member function 364
- operator-
 - TPoint member function 302
 - TRect member function 342
 - TSize member function 364
- operator=
 - TStatus data member 375
- operator |
 - TRect member function 342
- operator !
 - TPointer member function 305
- operator =
 - TCelArray member function 56
 - TPointer member function 305
 - TRegion member function 347
- operator ~
 - TBitSet member function 46
 - TPointer member function 305
- operator!=
 - TPoint member function 303
 - TRect member function 343
 - TSize member function 364
- operator&=
 - TRect member function 343
- operator+=
 - TPoint member function 303
 - TRect member function 343
 - TSize member function 364
- operator-=
 - TPoint member function 303
 - TRect member function 343
 - TSize member function 365
- operator<<
 - TPoint friend 303
 - TRect friend 344, 345
 - TResID friend 350, 351
 - TSize friend 365
- operator==
 - TColor member function 75
 - TPoint member function 303
 - TRect member function 343
 - TRegion member function 347
 - TSize member function 364
- operator>>
 - TPoint friend 303
 - TRect friend 344
 - TResID friend 350
 - TSize friend 365
- operator |=
 - TRect member function 343
- operator &=
 - TBitSet member function 46
 - TRegion member function 348
- operator +=
 - TBitSet member function 45
 - TRegion member function 347
- operator -=
 - TBitSet member function 46
 - TRegion member function 347
- operator []
 - TCelArray member function 56
- operator ^=
 - TRegion member function 348
- operator |=
 - TBitSet member function 46
 - TRegion member function 348
- operator BITMAPINFO()
 - TDib member function 149
- operator BITMAPINFOHEADER()
 - TDib member function 150
- operator COLORREF()
 - TColor member function 75
- operator delete
 - TPointer member function 305
- operator FARPROC
 - TProcInstance class 333
- operator HANDLE()
 - TDib member function 150
- operator HBITMAP()
 - TBitmap member function 42
- operator HDC()
 - TDC member function 116
- operator HDROP()
 - TDropInfo member function 171
- operator HINSTANCE
 - TModule member function 286

- operator HMETAFILE()
 - TMetaFilePict member function 280
- operator LPSTR()
 - TResID member function 350
- operator POINT*()
 - TPoint member function 303
- operator TBitmap&
 - TCelArray member function 56
- operator TPoint*()
 - TRect member function 343
- operator TRgbQuad()
 - TDib member function 150
- operators
 - TStatusBar class 377
- Options
 - TValidator data member 394
- OrgBitmap
 - TPaintDC data member 270
- OrgBrush
 - TDC data member 134
- OrgFont
 - TDC data member 135
- OrgPalette
 - TDC data member 135
- OrgPen
 - TDC data member 135
- OrgTextBrush
 - TDC data member 135
- Origin
 - TEditView data member 188
 - TListView data member 257
- OtherEdge
 - TLayoutConstraint data member 240
- OutStream
 - TDocument member function 166
 - TFileDocument member function 193
- OvertypemodeChange
 - TKeyboardModeTracker member function 239
- Overtypestate
 - TKeyboardModeTracker data member 238
- OWL\INCLUDE directory
 - and header files 11
- _OWLCLASS macro 29
- OWLCMD.CPP 384
- _OWLDATA macro 29

- OWLDialog
 - default ObjectWindows class
 - modeless dialog box 143
- _OWLDLL macro 29
- _OWLFAR macro 29
- _OWLFUNC macro 29
- OWLGetVersion member function 30

P

- PageMagnitude
 - TScrollBar data member 354
- PageNum
 - TPreviewPage data member 307
- PageSize
 - TPrintout data member 332
- pagination 331
 - page ranges 331
- Paint
 - TBitmapGadget member function 44
 - TButtonGadget member function 54
 - TGadget data member 214
 - TGadgetWindow member function 220
 - TGauge data member 224
 - TPreviewPage member function 307
 - TTextGadget member function 382
 - TWindow member function 434
- PAINT.CPP sample program 201
- PaintBorder
 - TGadget data member 214
- PaintButton
 - TTinyCaption member function 388
- PaintCaption
 - TTinyCaption member function 388
- PaintCloseBox
 - TTinyCaption member function 389
- PaintGadgets
 - TGadgetWindow member function 220
 - TMessageBar member function 277
- PaintInfo structure 434
- painting
 - horizontal rulers 233
 - slots 233
 - windows 207
- painting controls 54
- PaintMaxBox
 - TTinyCaption member function 389

- PaintMinBox
 - TTinyCaption member function 389
- PaintRgn
 - TDC member function 116
- PaintRuler
 - THSlider member function 233
 - TSlider member function 370
 - TVSlider member function 411
- PaintSlot
 - THSlider member function 233
 - TSlider member function 370
 - TVSlider member function 411
- PaintSysBox
 - TTinyCaption member function 389
- PaintThumb
 - TSlider member function 370
- PaintToPos
 - THSlider member function 233
- PalIndex
 - TColor member function 75
- PalRelative
 - TColor member function 75
- Param
 - TDialogAttr data member 144
 - TWindowAttr structure 450
- parameterized subclass
 - view and document classes 160
- Parent
 - TWindow data member 413
- parent windows 413
 - handles, returning 284
- passwords 176, 178
- Paste
 - TEdit member function 178
- PatBlt
 - TDC member function 116
- PathToRegion
 - TDC member function 117
- pd
 - TPrintDialog data member 320
- Percent constant 241
- PercentOf
 - TEdgeConstraint member function 172
 - TEdgeOrSizeConstraint member function 173
- PerformCreate
 - TEditView member function 187
 - TMDIChild member function 262
- TVbxControl member function 401
- TWindow member function 435
- pfxxxx property attribute constants 28
- Pic
 - TPXPictureValidator data member 335
- Picture
 - TPXPictureValidator member function 334
- picture strings
 - checking 333
 - scanning 335
 - valid characters 335
- picture validators 333
- Pie
 - TDC member function 117
- pixels to twips conversion 407
- placing gadgets 301
- Planes
 - TBitmap member function 43
- PlayMetaFile
 - TDC member function 118
- PlayMetaFileRecord
 - TDC member function 118
- PlgBlt
 - TDC member function 118
- Pmf
 - TResponseTableEntry data member 351
- pointer to a member function 352
 - and message dispatchers 465
- pointers
 - member functions 31
 - transfer buffers 80
- PointSize
 - TChooseFontDialog::TData data member 66
- PointToPos
 - TSlider member function 370
 - TVSlider member function 411
- PolyBezier
 - TDC member function 119
- PolyBezierTo
 - TDC member function 119
- PolyDraw
 - TDC member function 119
- Polygon
 - TDC member function 120
- Polyline
 - TDC member function 120

- PolylineTo
 - TDC member function 121
- PolyPolygon
 - TDC member function 121
- PolyPolyline
 - TDC member function 121
- pop-up menus
 - creating 305
- Pos
 - TSlider data member 371
- Position
 - TScrollBarData data member 358
- position
 - caret, returning 176, 178
 - character, edit control 174, 176
 - current 178, 442
 - coordinates 355
 - list box 249
 - moving 356
 - text selection 177, 251, 252, 253
 - edit controls
 - tab stops in 179
 - menu (MDI) 267
 - relative to origin 49, 76, 84, 174, 231, 248, 336, 373
 - scroll bar
 - thumb 354, 360, 361
 - range 355
 - setting 356
 - tracking 356
 - scroller 360
 - specified by variables 175, 179
- PositionGadget
 - TControlBar member function 87
 - TGadgetWindow member function 221
 - TStatusBar member function 378
- PostDispatchAction
 - TApplication member function 35
- PostDocError
 - TDocManager member function 155
- PostError
 - TDocument member function 166
- PostMessage
 - TWindow member function 434
- PosToPoint
 - THSlider member function 233
 - TSlider member function 370
 - TVSlider member function 412
- prComplete constant 334
- PreProcessMenu
 - TApplication member function 35
- PreProcessMsg
 - TControlBar member function 87
 - TDecoratedFrame member function 137
 - TDialoag member function 142
 - TFrameWindow member function 205
 - TMDIChild member function 262
 - TMDIClient member function 265
 - TWindow member function 435
- prError constant 334
- previewing data 323
- previewing pages 306
- Previous
 - TWindow member function 435
- prInComplete constant 334
- Print
 - TPrinter member function 328
- print preview classes 306, 323
- print setup dialog boxes
 - controls, initializing 320
 - creating 319, 321
- PrintDC
 - TPreviewPage data member 307
- printer banding flags 332
- printers 327
 - changing 327, 329
 - configuring 319, 328
 - default
 - returning 319, 327
 - updating 329
 - device
 - changing 329
 - clearing 328
 - errors 321
 - reporting 328
 - IDs
 - dialog and control 23
 - string constants 25
 - settings, initializing 320, 321
 - status, determining 25
- PrintExtent
 - TPreviewPage data member 307
- printing 319, 330
 - device handle 332

- discontinuing 328, 329
- errors 321
 - reporting 328
- events, responding to 320
- jobs, labeling 329, 330, 332
- multiple pages 331
- selected pages 331
- specifications
 - copying 319
 - initializing 320, 321
 - page size 332
- Printout
 - TPreviewPage data member 307
- printouts, banding 331, 332
- PrintPage
 - TPrintout member function 331
- PrnDC
 - TPrintPreviewDC data member 327
- PrnFont
 - TPrintPreviewDC data member 327
- Process
 - TPXPictureValidator member function 335
- ProcessAppMsg
 - TApplication member function 36
- procinstance 333
- program example
 - custom cursor 439
- programs
 - displaying current state 376, 377
- prompt
 - TInputDialog data member 235
- property attributes
 - pfxxx constants 28
- Property enum
 - TDocument data member 163
 - TView data member 408
- property index
 - and TFileDocument 193
- property list
 - retrieving handles 428
- PropertyCount
 - TDocument member function 166
 - TView member function 409
- PropertyFlags
 - TDocument member function 167
 - TFileDocument member function 193
 - TView member function 410
- PropertyName
 - TDocument member function 167
 - TFileDocument member function 194
 - TView member function 410
- protected constructor
 - TStream class 379
- protected data member
 - TFileDocument class 194
- protected data members
 - TApplication class 37
 - TButton class 49
 - TCelArray class 57
 - TChooseColorDialog class 61
 - TChooseFontDialog class 64
 - TClipboard class 70
 - TClipboardViewer class 72
 - TCommonDialog class 82
 - TDecoratedFrame class 137
 - TDocument class 168
 - TEdit class 179
 - TEditView class 188
 - TFilterValidator class 197
 - TFindReplaceDialog class 198
 - TFrameWindow class 206
 - TLayoutWindow class 248
 - TListView class 257
 - TMenu class 273
 - TModule class 287
 - TOpenSaveDialog class 288
 - TPreviewPage class 307
 - TPrintDialog class 320
 - TPrinter class 328
 - TPrintout class 332
 - TPrintPreviewDC class 327
 - TPXPictureValidator class 335
 - TRangeValidator class 338
 - TSlider class 371
 - TStatusBar class 377
 - TStream class 379
 - TStringLookupValidator class 380
 - TTinyCaption class 384
 - TValidator class 394
 - TView class 410
 - TWindow class 445
- protected member functions
 - TApplication class 38
 - TBitmapGadget class 44

- TButton class 49
- TCheckBox class 60
- TChooseColorDialog class 62
- TChooseFontDialog class 64
- TComboBox class 80
- TCommonDialog class 83
- TControl class 85
- TDecoratedFrame class 137
- TDecoratedMDIFrame class 138
- TDialog class 143
- TDocManager class 156
- TDocTemplate class 160
- TDocument class 168
- TEdit class 179
- TEditFile class 184
- TEditView class 188
- TEventHandler class 190
- TFileDocument class 194
- TFindDialog class 197
- TFindReplaceDialog class 199
- TFrameWindow class 206
- TGroupBox class 232
- TInputDialog class 236
- TLayoutWindow class 248
- TListBox class 254
- TListView class 257
- TMDIChild class 262
- TMDIClient class 265
- TMDIFrame class 268
- TOpenSaveDialog 289
- TPreviewPage class 308
- TPrinter class 329
- TPrintPreviewDC class 327
- TPXPictureValidator class 335
- TRadioButton class 337
- TReplaceDialog class 349
- TScrollBar class 357
- TSlider class 368
- TStatic class 375
- TStatusBar class 378
- TTinyCaption class 385
- TVbxControl class 401
- TVbxEventHandler class 407
- TView class 410
- TWindow class 446
- prXxxx constants 300

- Ps
 - TPaintDC data member 292
- PtIn
 - TGadget data member 214
- PtVisible
 - TDC member function 121
- public data member
 - TListView class 256
- public data members
 - TApplication class 31
 - TButton class 49
 - TCheckBox class 58
 - TChooseColorDialog::TData struct 63
 - TChooseFontDialog::TData struct 65
 - TClipboard class 67
 - TComboBox class 76
 - TComboBoxData class 81
 - TDialog class 140
 - TDocManager class 153
 - TDocument class 163
 - TEditFile class 182
 - TEditSearch class 185
 - TEventInfo class 190
 - TFloatingFrame class 201
 - TFrameWindow class 204
 - TGroupBox class 231
 - TInputDialog class 235
 - TLayoutConstraint structure 240
 - TLayoutMetrics class 242
 - TListBoxData structure 254
 - TMDIClient class 263
 - TMDIFrame class 267
 - TModule class 280
 - TOpenSaveDialog::TData 290
 - TPrintDialog::TData struct 321
 - TResponseTableEntry class 351
 - TScrollBar class 354
 - TScrollBarData structure 358
 - TStatic class 373
 - TStatus class 375
 - TStatusBar class 376
 - TStream class 378
 - TView class 408
 - TWindow class 413
 - TWindowAttr structure 449
- public destructor
 - TStream class 378

- public member functions
 - TApplication class 33
 - TBitmapGadget class 44
 - TBitSet class 45
 - TCelArray class 56
 - TCharSet class 58
 - TCheckBox class 59
 - TChooseColorDialog class 61
 - TClipboard class 67
 - TClipboardViewer class 71
 - TComboBox class 77
 - TComboBoxData class 81
 - TCommonDialog class 82
 - TControl class 84
 - TDecoratedFrame class 136
 - TDialog class 140
 - TDocManager class 153
 - TDocTemplate class 157
 - TDocTemplateT class 161
 - TDocument class 164
 - TEdgeConstraint structure 171
 - TEdgeOrSizeConstraint structure 172
 - TEdit class 174
 - TEditFile class 183
 - TEditSearch class 186
 - TEditView class 187
 - TFileDocument class 192
 - TFileOpenDialog class 195
 - TFileSaveDialog class 196
 - TFilterValidator class 196
 - TFindReplaceDialog::TData struct 199
 - TFindReplaceDialog class 198
 - TFrameWindow class 204
 - TGroupBox class 231
 - TInputDialog class 236
 - TLayoutWindow class 247
 - TListBox class 249
 - TListBoxData structure 255
 - TListView class 256
 - TLookupValidator class 260
 - TMDIChild class 261
 - TMDIClient class 263
 - TMDIFrame class 268
 - TMenu class 271
 - TModule class 282
 - TOpenSaveDialog class 288
 - TPointer class 304
 - TPopupMenu class 305
 - TPreviewPage class 307
 - TPrintDialog::TData struct 322
 - TPrintDialog class 319
 - TPrinter class 328
 - TPrinterAbortDlg class 330
 - TPrintout class 330
 - TPrintPreviewDC class 324
 - TPXPictureValidator class 334
 - TRangeValidator class 337
 - TScrollBar class 355
 - TScroller class 360
 - TSlider class 367
 - TStatic class 374
 - TStatusBar class 376
 - TStream class 379
 - TStringLookupValidator class 380
 - TValidator class 392
 - TVbxControl class 397
 - TView class 409
 - TWindow class 415
 - TWindowView class 451
- pull-down menu items
 - defining 276
- PumpWaitingMessages
 - TApplication member function 36
- pure virtual
 - Create function 417
 - registration GetClassName function 446
- push buttons 48
 - default 48

Q

- QueryAbort
 - TPrintDC member function 317
- QueryCreate
 - TClipboard member function 69
- QueryEscSupport
 - TPrintDC member function 317
- QueryLink
 - TClipboard member function 69
- QueryThrow
 - TApplication member function 36
- QueryViews
 - TDocument member function 167

R

radio buttons 336

Range

 TSlider data member 372

range validators 337

range values

 converting to scroll values 26

ranges

 numeric values

 testing for 337

Read

 TDib member function 151

 TEditFile member function 184

RealizePalette

 TDC member function 121

ReceiveMessage

 TWindow member function 435

Rectangle

 TDC member function 122

RectVisible

 TDC member function 122

Red

 TColor member function 75

RedrawWindow

 TWindow member function 435

RefAdd

 TGdiObject member function 226

RefCount

 TGdiObject member function 226

RefDec

 TGdiObject member functions 226

RefFind

 TGdiObject member function 226

RefInc

 TGdiObject member function 227

RefRemove

 TGdiObject member function 227

Refresh

 TVbxControl member function 400

RefTemplate

 TDocManager member function 156

Register

 TWindow member function 435

RegisterClipboardFormat

 TClipboard member function 70

RegisterHotKey

 TWindow member function 435

registration

 Windows 435

relational databases

 validity checking 333

Relationship

 TLayoutConstraint data member 240

RelWin

 TLayoutConstraint data member 240

Remove

 TDocument::List member function 169

 TGadgetWindow member function 217

RemoveChild

 TWindow member function 446

RemoveChildLayoutMetrics

 TLayoutWindow member function 247

Removed

 TControlGadget member function 88

 TGadget member function 214

RemoveItem

 TVbxControl member function 400

RemoveMenu

 TMenu member function 273

RemoveProp

 TWindow member function 436

removing a property 436

ReOrg

 TPrintPreviewDC member function 324

ReplaceWith

 TEditFile member function 184

 TFindReplaceDialog::TData member function 200

ReportError

 TPrinter member function 328

ReScale

 TPrintPreviewDC member function 324

ResetDC

 TDC member function 122

ResetSelections

 TListBoxData member function 255

ResizePalette

 TPalette member function 296

resource files

 described 15

resource ID

 TXOwl 454

resource IDs

 retrieving default 142

- ResourceIdToString
 - TXOwl public member function 454
- resources
 - callback functions and 286
 - finding 282, 283
 - handles, returning 285
 - loading 286
 - into memory 282, 285
 - Windows applications 286
 - size, returning 286
- response functions 457
- response table
 - TFloatingFrame class 202
- response table entries 351
 - TWindowView 452
- Response table entry
 - TListView member functions 259
- response table macro
 - and signature template 465
- response table macros
 - for VBX events 404
- response tables
 - and document manager 152
 - declaring 18
 - defining 19, 21
 - TButton class 50
 - TCheckBox class 61
 - TChooseColorDialog class 62
 - TChooseFontDialog class 65
 - TClipboardViewer class 72
 - TCommonDialog class 83
 - TControl class 86
 - TControlGadget 89
 - TDecoratedFrame class 138
 - TDecoratedMDIFrame class 139
 - TDialog class 144
 - TDocManager class 157
 - TEdit class 181
 - TEditFile class 185
 - TEditSearch class 186
 - TFrameWindow class 208
 - TKeyboardModeTracker class 239
 - TLayoutWindow class 248
 - TMDIChild class 263
 - TMDIClient class 266
 - TMDIFrame class 268
 - TOpenSaveDialog class 290
 - TPreviewPage 308
 - TPrintDialog class 320
 - TRadioButton class 337
 - TSlider class 372
 - TTinyCaption class 389
 - TVbxControl 402
 - TVbxEventHandler 407
 - TWindow class 447
- RestoreBitmap
 - TMemoryDC 269
- RestoreBrush
 - TDC member function 122
- RestoreDC
 - TDC member function 122
- RestoreFont
 - TDC member function 122
 - TPrintPreviewDC member function 324
- RestoreMemory
 - TModule member function 286
- RestoreMenu
 - TFrameWindow member function 205
- RestoreObjects
 - TDC member function 122
 - TMemoryDC 269
- RestorePalette
 - TDC member function 123
- RestorePen
 - TDC member function 123
- RestoreTextBrush
 - TDC member function 123
- ResumeThrow
 - TApplication member function 36
- retrieving information about events 190
- returning a pointer to a TWindow 428
- Revert
 - TDocument member function 167
 - TFileDocument member function 194
- Rgb
 - TColor member function 75
- RightOf
 - TEdgeConstraint member function 172
- RootDocument
 - TDocument member function 167
- RoundRect
 - TDC member function 123
- Run
 - TApplication member function 36

S

SameAs

- TEdgeConstraint member function 172
- TEdgeOrSizeConstraint member function 173

sample class entry 8

sample programs

- caption bars 384
- sliders 366
- VALIDATE.CPP 391

Save

- TEditFile member function 184

SaveAs

- TEditFile member function 184

SaveDC

- TDC member function 123

SB_ scroll bar constants 360, 361

- SB_BOTTOM constant 356
 - SB_LINEDOWN constant 356
 - SB_LINEUP constant 356
 - SB_PAGEDOWN constant 356
 - SB_PAGEUP constant 356
 - SB_THUMBPOSITION constant 356
 - SB_THUMBTRACK constant 356
 - SB_TOP constant 357
- ### SBBottom

- TScrollBar member function 355

SBLineDown

- TScrollBar member function 356

SBLineUp

- TScrollBar member function 356

SBPageDown

- TScrollBar member function 356

SBPageUp

- TScrollBar member function 356

SBS_HORZ constant 355

SBS_VERT constant 355

SBThumbPosition

- TScrollBar member function 356

SBThumbTrack

- TScrollBar member function 356

SBTop

- TScrollBar member function 356

ScaleViewportExt

- TDC member function 123
- TPrintPreviewDC member function 325

ScaleWindowExt

- TDC member function 123

- TPrintPreviewDC member function 325

Scan

- TPXPictureValidator member function 335

scanning picture formats 335

scope resolution operator

- Windows API calls 3

screen device

- logical point conversions 324, 325

screen resolution

- layout units and 241

ScreenToClient

- TWindow member function 436

Scroll

- TEdit member function 178

scroll bars 354, 358

position

- line down 356
- thumb 354, 356, 360, 361
- returning 358

range

- getting 355
- setting 361
- thumb 354, 355

sliders vs. 365

transferring 357

TScrollBarData 358

warning 354

scroll values

- converting to range values 26

ScrollBy

- TScroller member function 361

ScrollDC

- TDC member function 124

Scroller

- TWindow data member 413

scroller 413

scrolling windows

page size

- setting 360

scroll bars

- horizontal 360
- position 360
- range, setting 361
- vertical 361

units

- setting 361

- ScrollLockModeChange
 - TKeyboardModeTracker member function 239
- ScrollLockState
 - TKeyboardModeTracker data member 238
- ScrollTo
 - TScroller member function 361
- ScrollWindow
 - TWindow member function 436
- ScrollWindowEx
 - TWindow member function 436
- SD_INPUTDIALOG constant 236
- SDPtLP
 - TPrintPreviewDC member function 325
- search 185
 - case-sensitive or case-insensitive 178
 - list box 249
- Search, TEdit member function 178
- SearchCmd
 - TEditSearch data member 185
- SearchData
 - TEditSearch data member 185
- SearchDialog
 - TEditSearch data member 185
- searching for
 - specific strings 260, 380
 - specified characters 335, 336
- SelCount
 - TListBoxData data member 254
- Select
 - TListBoxData member function 255
- select and restore functions 324, 325
- SelectAnySave
 - TDocManager member function 156
- SelectClipPath
 - TDC member function 124
- SelectClipRgn
 - TDC member function 125
- SelectDocType
 - TDocManager member function 156
- SelectImage
 - TBitmapGadget member function 44
- Selection
 - TComboBoxData data member 81
- selection 230
 - adding strings 254, 255
 - colors 61
 - data transfer 254, 255
 - fonts 63
 - number of items 254
 - removing strings 254, 255
 - text 177, 179
- SelectionChanged
 - TGroupBox member function 231
- SelectObject
 - TDC member function 125
 - TMemoryDC 270
 - TPrintPreviewDC member function 325
- SelectSave
 - TDocTemplate member function 159
- SelectStockObject
 - TDC member function 125
 - TPrintPreviewDC member function 325
- SelectString
 - TListBoxData member function 255
- SelectViewType
 - TDocManager member function 157
- SelIndex
 - TComboBoxData data member 81
- SelStrings
 - TListBoxData data member 254
- SendDlgItemMessage
 - TWindow member function 437
- SendDlgItemMsg
 - TDialog member function 143
- sending messages 416
- SendMessage
 - TWindow member function 437
- SendNotification
 - TWindow member function 437
- separators
 - width and height 362
- Set
 - TEdgeConstraint member function 172
 - TRect member function 344
- SetAbortProc
 - TPrintDC member function 317
- SetActiveWindow
 - TWindow member function 437
- SetBitmapBits
 - TBitmap member function 43
- SetBitmapDimension
 - TBitmap member function 43
- SetBkColor
 - TDC member function 125

TPrintPreviewDC member function 326
 SetBkgndColor
 TWindow member function 437
 SetBkMode
 TDC member function 126
 SetBorders
 TGadget data member 211
 SetBorderStyle
 TGadget data member 211
 SetBounds
 TBitmapGadget member function 44
 TButtonGadget member function 55
 TControlGadget member function 88
 TGadget data member 211
 SetBoundsRect
 TDC member function 126
 SetBrushOrg
 TDC member function 126
 SetButtonState
 TButtonGadget member function 52
 SetButtonType
 TButtonGadget member function 52
 SetCaption
 TDialog member function 143
 TWindow member function 438
 SetCaretBlinkTime
 TWindow member function 438
 SetCaretIndex
 TListBox member function 252
 SetCaretPos
 TWindow member function 438
 SetCelSize
 TCelArray member function 56
 SetCheck
 TCheckBox member function 59
 SetChildLayoutMetrics
 TLayoutWindow member function 247
 SetClassLong
 TWindow member function 438
 SetClassWord
 TWindow member function 438
 SetClientWindow
 TFrameWindow member function 205
 SetClipboardData
 TClipboard member function 70
 SetClipboardViewer
 TClipboard member function 70
 SetColor
 TDib member function 150
 SetColumnWidth
 TListBox member function 252
 SetCopyCount
 TPrintDC member function 318
 SetCursor
 TWindow member function 438
 SetDefaultExt
 TDocTemplate member function 159
 SetDefaultId
 TDialog member function 143
 SetDevMode
 TPrintDialog::TData member function 323
 SetDevNames
 TPrintDialog::TData member function 323
 SetDIBits
 TDC member function 126
 SetDIBitsToDevice
 TDC member function 127
 SetDirection
 TGadgetWindow member function 217
 TToolBox data member 390
 SetDirectory
 TDocTemplate member function 159
 SetDlgItem
 TWindow member function 439
 SetDlgItemText
 TWindow member function 439
 SetDocManager
 TApplication member function 36
 TDocTemplate member function 159
 SetDocmanager
 TDocument member function 167
 SetDocPath
 TDocument member function 167
 SetDocTitle
 TEditView member function 188
 TListView member function 257
 TView member function 410
 TWindow member function 439
 TWindowView member function 452
 SetEditSel
 TComboBox member function 79
 SetEmpty
 TRect member function 344

- SetEnabled
 - TGadget data member 211
- SetExtendedUI
 - TComboBox member function 79
- SetExtent
 - TListView member functions 259
- SetFileFilter
 - TDocTemplate member function 159
- SetFileName
 - TEditFile member function 184
- SetFilter
 - TOpenSaveDialog::TData struct 292
- SetFlag
 - TDocTemplate member function 159
 - TWindow member function 439
- SetHandle
 - TEdit member function 178
- SetHintCommand
 - TGadgetWindow member function 217
- SetHintMode
 - TGadgetWindow member function 217
- SetHintText
 - TMessageBar member function 276
- SetHorizontalExtent
 - TListBox member function 252
- SetIcon
 - TFrameWindow member function 206
- SetIndex
 - TDib member function 150
- SetInstance
 - TModule member function 286
- SetItemData
 - TComboBox member function 79
 - TListBox member function 252
- SetItemHeight
 - TComboBox member function 79
 - TListBox member function 252
- SetLed
 - TGauge data member 222
- SetMainWindow
 - TApplication member function 36
- SetMainWinParams
 - TApplication member function 37
- SetMapMode
 - TDC member function 127
 - TPrintPreviewDC member function 326
- SetMapperFlags
 - TDC member function 127
- SetMargins
 - TFloatingFrame member functions 201
 - TGadget data member 211
 - TGadgetWindow member function 217
- SetMenu
 - TMDIFrame member function 268
 - TWindow member function 439
- SetMenuDescr
 - TFrameWindow member function 206
- SetMenuItemBitmaps
 - TMenu member function 273
- SetMiterLimit
 - TDC member function 128
- SetModeIndicator
 - TStatusBar member function 377
- SetModule
 - TWindow member function 439
- SetNotchCorners
 - TButtonGadget member function 52
- SetNumCels
 - TCelArray member function 56
- SetOffset
 - TCelArray member function 56
- SetOpenMode
 - TDocument member function 167
- SetOption
 - TValidator member function 393
- SetPageNumber
 - TPreviewPage member function 307
- SetPageSize
 - TScroller member function 360
- SetPaletteEntries
 - TPalette member function 296
- SetPaletteEntry
 - TPalette member function 296
- SetParent
 - TWindow member function 440
- SetPasswordChar
 - TEdit member function 178
- SetPixel
 - TDC member function 128
- SetPolyFillMode
 - TDC member function 128
- SetPosition
 - TScrollBar member function 357

- TSlider member function 368
- SetPrinter
 - TPrinter member function 329
- SetPrintParams
 - TPrintout member function 331
- SetProp
 - TVbxControl member function 400, 401
 - TWindow member function 440
- SetProperty
 - TDocument member function 168
 - TFileDocument member function 194
 - TView member function 410
- SetRange
 - TGauge data member 222
 - TScrollBar member function 357
 - TScroller member function 361
 - TSlider member function 368
- SetReadOnly
 - TEdit member function 178
- SetRect
 - TEdit member function 178
- SetRectNP
 - TEdit member function 178
- SetRectRgn
 - TRegion member function 348
- SetResourceHandler
 - TModule member function 286
- SetRGBColor
 - TChooseColorDialog member function 61
- SetRGBMsgId
 - TChooseColorDialog data member 62
- SetROP2
 - TDC member function 128
- SetRuler
 - TSlider member function 368
- SetSBarRange
 - TScroller member function 361
- SetScrollPos
 - TWindow member function 440
- SetScrollRange
 - TWindow member function 440
- SetSel
 - TListBox member function 252
- SetSelection
 - TEdit member function 179
- SetSelIndex
 - TComboBox member function 79
 - TListBox member function 252
- SetSelIndexes
 - TListBox member function 252
- SetSelItemRange
 - TListBox member function 253
- SetSelString
 - TComboBox member function 79
 - TListBox member function 253
- SetSelStrings
 - TListBox member function 253
- SetShqdownStyle
 - TButtonGadget member function 52
- SetShrinkWrap
 - TGadget member function 211
 - TGadgetWindow member function 217
- SetSize
 - TGadget data member 211
- SetSpacing
 - TStatusBar member function 377
- SetState
 - TCheckBox member function 59
- SetStretchBltMode
 - TDC member function 129
- SetStyle
 - TCheckBox member function 59
- SetSysModalWindow
 - TWindow member function 440
- SetSystemPaletteUse
 - TDC member function 129
- SetTabStops
 - TEdit member function 179
 - TListBox member function 253
- SetTemplate
 - TDocument member function 168
- SetText
 - TComboBox member function 80
 - TMessageBar member function 276
 - TStatic member function 374
 - TTextGadget member function 381
- SetTextAlign
 - TDC member function 129
- SetTextCharacterExtra
 - TDC member function 129
- SetTextColor
 - TDC member function 130
 - TPrintPreviewDC member function 326

- SetTextJustification
 - TDC member function 130
- SetTimer
 - TWindow member function 441
- setting bits 45
- SetTitle
 - TDocument member function 168
- SetTopIndex
 - TListBox member function 253
- SetTransferBuffer
 - TWindow member function 441
- SetUnits
 - TScroller member function 361
- Setup
 - TPrinter member function 328
- SetupThumbRgn
 - TSlider member function 371
- SetupWindow
 - TButton member function 50
 - TClipboardViewer member function 71
 - TComboBox member function 80
 - TCommonDialog member function 83
 - TDecoratedFrame member function 138
 - TDialog member function 144
 - TEdit member function 181
 - TEditFile member function 184
 - TEditSearch member function 186
 - TFrameWindow member function 207
 - TInputDialog member function 236
 - TPrinterAbortDlg member function 330
 - TScrollBar member function 357
 - TSlider member function 371
 - TWindow member function 447
- SetValue
 - TGauge data member 222
- SetViewMenu
 - TView member function 410
- SetViewportExt
 - TDC member function 130
 - TPrintPreviewDC member function 326
- SetViewportOrg
 - TDC member function 130
 - TPrintPreviewDC member function 326
- SetWindow
 - TScroller member function 361
- SetWindowExt
 - TDC member function 131
- TPrintPreviewDC member function 326
- SetWindowLong
 - TWindow member function 441
- SetWindowOrg
 - TDC member function 131
- SetWindowPlacement
 - TWindow member function 441
- SetWindowPos
 - TWindow member function 441
- SetWindowText
 - TWindow member function 442
- SetWindowWord
 - TWindow member function 442
- SetWordBreakProc
 - TEdit member function 179
- SetWorldTransform
 - TDC member function 131
- ShareViMsgId
 - TOpenSaveDialog data member 288
- ShareViolation
 - TOpenSaveDialog member function 289
- ShouldDelete
 - TCelArray data member 57
 - TDC data member 135
 - TGdiObject data member 227
 - TMenu data member 273
- Show
 - TWindow member function 442
- ShowCaret
 - TWindow member function 442
- showCmd constants 442
- ShowList
 - TComboBox member function 80
- ShowOwnedPopups
 - TWindow member function 442
- ShowScrollBar
 - TWindow member function 443
- ShowWindow
 - TWindow member function 443
- shrink-wrapping for gadgets 215
- shrinkToClient
 - in TFrameWindow's constructor 204
- shrinktoClient
 - and notify parent window event 207
- ShrinkWrapHeight
 - TGadget data member 212
 - TGadgetWindow data member 219

- ShrinkWrapWidth
 - TGadget data member 212
 - TGadgetWindow data member 219
- shxxxx document sharing enum (constants) 30
- signature abbreviations
 - and data types 466
 - used by member functions 466
- signature template
 - used in response table macro 465
- Size
 - TDib member function 150
 - TRect member function 344
- size constraints, creating windows 241
- SizeMax
 - TChooseFontDialog::TData data member 66
- SizeMin
 - TChooseFontDialog::TData data member 66
- SizeOfResource
 - TModule member function 286
- SkipToComma
 - TPXPictureValidator member function 336
- slicing bitmaps 55
- SlideDc
 - TSlider data member 372
- SLIDER.CPP 366
- SLIDER.CPP sample program 367
- sliders
 - background, erasing 368
 - background colors 370
 - storing 371
 - description of 365
 - horizontal 232
 - objects
 - constructing 367
 - destructing 367
 - painting 368, 370
 - entire 370
 - rulers in 370
 - thumbs 370
 - recalculating sizes 370
 - resource ID
 - thumb knob 372
 - thumb positions
 - aligning with tick positions 366, 368
 - current position 367
 - moving 368, 371
 - present range 367
 - returning 372
 - setting 369
 - snapping 371, 372
 - translating 370, 371
 - thumb shape, defining 371, 372
 - tick positions and 366
 - setting gaps 372
 - vertical 411
- SlideThumb
 - TSlider member function 371
- Sliding
 - TSlider data member 372
- SlotThick
 - TSlider data member 372
- Snap
 - TSlider data member 372
- snapping 371, 372
- SnapPos
 - TSlider member function 371
- sounds, beep 180
- Spacing
 - TStatusBar data member 378
- SS_LEFT constant 374
- StartDoc
 - TPrintDC member function 318
- StartScan
 - TDib member function 150
- static controls
 - registration class name 375
 - resources and associating with objects 374
 - text length 373
- StaticName
 - TEditView member function 188
 - TListView member function 257
 - TWindowView member function 452
- Status
 - TModule data member 281
 - TWindow data member 413
- status, main window 281
- status bars 375
 - borders 377
 - creating 376
 - inserting text in 376, 377
 - spacing items in 377, 378
- status code
 - TXCompatibility constructor 453
- status lines 136

- StockId
 - TBrush data member 46
- Stocks[]
 - TBrush data member 47
 - TFont data member 202
 - TPalette data member 293
 - TPen data member 299
- streaming
 - TFileDocument 193
 - TInStream 236
 - TOutStream 292
- StreamName
 - TStream data member 378
- streams
 - constructing
 - document modes 27
 - open 193
- StretchBlt
 - TDC member function 131
- StretchDIBits
 - TDC member function 132
- string-lookup validators 379
- string message
 - TXOwl 454
- Strings
 - TComboBoxData data member 81
 - TListBoxData data member 254
 - TStringLookupValidator data member 380
- strings
 - adding to
 - list boxes 249, 254, 255
 - checking validity of 379
 - comparing 196, 249, 260
 - picture 333
 - concatenating 255
 - deleting 249
 - finding 255
 - getting 251
 - ID constants 23
 - list view 25
 - inserting 252
 - length 373
 - getting 251
 - loading into memory 286
 - searching for 260, 380
- StrokeAndFillPath
 - TDC member function 132

- StrokePath
 - TDC member function 132
- structs
 - BANDINFOSTRUCT 40
 - DOCINFO 319
 - TBandInfo 40
- structures
 - TWindowAttr 449
- Style
 - TChooseFontDialog::TData data member 66
 - TWindowAttr structure 450
- styles
 - button 49
 - combo boxes 76
 - edit controls 174
 - list box 248
 - scroll bars 354
- SubclassWindowFunction
 - TWindow member function 443
- SuspendThrow
 - TApplication member function 37
- SyncFont
 - TPrintPreviewDC member function 327
- synchronizing functions 324, 327
- SyntaxCheck
 - TPXPictureValidator member function 336
- SysColorChange
 - TBitmapGadget member function 44
 - TButtonGadget member function 52
- SyscolorChange
 - TGadget member function 212
- system menu, creating 380

T

- T
 - TResponseTableEntry member function 352
- tab stops, creating 231
- tab stops, edit controls, setting 179
- TabbedTextOut
 - TDC member function 132
- TActionFunc typedef 30
- TActionMemFunc typedef 30
- Tag
 - TDocument data member 163
 - TView data member 408
- TAlign
 - TTextGadget data member 381

- TAnyDispatcher typedef *31*
- TAnyPMF typedef *31*
- TApplication
 - inheritance diagram *479*
- TApplication class *31*
 - constructors *32, 33*
 - data members
 - protected *37*
 - public *31*
 - destructor *33*
 - member functions
 - protected *38*
 - public *33*
- TAutoDelete
 - TDC data member *92*
 - TGdiObject data member *225*
- TBandInfo
 - struct *40*
- TButtonGadget
 - inheritance diagram *482*
- TBitmap
 - inheritance diagram *480*
- TBitmap
 - constructors *40*
 - member functions
 - GetBitmapBits *42*
 - GetBitmapDimension *42*
 - GetObject *42*
 - HandleCreate *43*
 - operator HBITMAP() *42*
 - SetBitmapBits *43*
 - SetBitmapDimension *43*
 - ToClipboard *43*
 - related operator
 - << *42*
- TBitmap* Bitmap
 - TCelArray data member *57*
- TBitmap member functions
 - BitsPixel *42*
 - Height *42*
 - Planes *43*
 - Width *43*
- TBitMapGadget
 - inheritance diagram *480*
- TBitmapGadget class *43*
 - constructor *44*
 - destructor *44*
 - member functions
 - protected *44*
 - public *44*
- TBitSet class *45*
 - constructors *45*
 - member functions
 - public *45*
- TBorders struct
 - TGadget data member *208*
- TBorderStyle
 - TGadget data member *209, 212*
- TBrush *46*
 - constructors *47*
 - data members
 - StockId *46*
 - Stocks[] *47*
 - member functions
 - GetObject *48*
 - GetStock *48*
 - operator HBRUSH() *48*
 - UnrealizeObject *48*
- TButton
 - inheritance diagram *481*
- TButton class *48*
 - constructors *49*
 - data members
 - protected *49*
 - public *49*
 - macros *50*
 - member functions, protected *49*
 - response table *50*
- TCelArray class *55*
 - constructors *55*
 - data members, protected *57*
 - destructors *55*
 - member functions, public *56*
- TCEnabled
 - TTinyCaption data member *384*
- TCharSet class *57*
 - constructors *57*
 - member functions
 - public *58*
- TCheckBox
 - inheritance diagram *483*
- TCheckBox class *58*
 - constructors *58*
 - data members, public *58*

- member functions
 - protected 60
 - public 59
- response table 61
- TChooseColorDialog
 - inheritance diagram 484
- TChooseColorDialog class 61
 - constructor 61
 - data members, protected 61
 - macro 62
 - member functions
 - protected 62
 - public 61
 - response table 62
 - TData structure 62
- TChooseFontDialog
 - inheritance diagram 486
- TChooseFontDialog::TData struct
 - data members
 - public 65
- TChooseFontDialog class 63
 - constructor 64
 - data members, protected 64
 - macro 65
 - member functions, protected 64
 - response table 65
 - TData structure 65
- TClientDC
 - constructor 67
- TClipboard class 67
 - constructor 70
 - data members
 - protected 70
 - public 67
 - destructor 70
 - member functions, public 67
 - operator BOOL 69
- TClipboardViewer
 - inheritance diagram 487
- TClipboardViewer class 71
 - constructor 71
 - data members, protected 72
 - macro 72
 - member functions, public 71
 - response table 72
- TColor
 - class 72
 - constructors 73
- TColor data members
 - Black 72
 - Gray 72
 - LtBlue 72
 - LtCyan 72
 - LtGray 72
 - LtGreen 73
 - LtMagenta 73
 - LtRed 73
 - LtYellow 73
 - Value 75
 - White 73
- TColor member functions
 - Blue 74
 - Flags 74
 - Green 74
 - Index 75
 - operator== 75
 - operator COLORREF() 75
 - PalIndex 75
 - PalRelative 75
 - Red 75
 - Rgb 75
- TComboBox
 - inheritance diagram 488
- TComboBox class 76
 - constructors 76, 77
 - data members, public 76
 - member functions
 - protected 80
 - public 77
- TComboBoxData
 - inheritance diagram 490
- TComboBoxData class 81
 - constructor 81
 - data members, public 81
 - destructor 81
 - member functions, public 81
- TCommandEnabler 422
- TCommonDialog
 - inheritance diagram 492
- TCommonDialog class 82
 - constructor 82
 - data members, protected 82
 - member functions
 - protected 83

- public 82
- response table 83
- TCondFunc type 83
- TCondMemFunc typedef 84
- TControl
 - inheritance diagram 493
- TControl class 84
 - constructors 84
 - member functions
 - protected 85
 - public 84
 - response table 86
- TControlBar
 - inheritance diagram 494
- TControlGadget
 - inheritance diagram 495
- TControlGadget class 87
 - constructors 88
 - destructor 88
 - response tables 89
- TCreatedDC
 - constructors 89
 - destructor 89
 - member function 90
- TCursor
 - constructors 90
 - destructor 91
 - member functions
 - GetIconInfo 91
 - HCURSOR[] 91
- td transfer function constants
 - TComboBox::Transfer and 80
- TData structure
 - TChooseColorDialog class 62
 - TChooseFontDialog class 65
 - TPrintDialog class 320
- TDC
 - constructors 92, 135
 - destructor 93
- TDC data members
 - Handle 134
 - OrgBrush 134
 - OrgFont 135
 - OrgPalette 135
 - OrgPen 135
 - OrgTextBrush 135
 - ShouldDelete 135
- TAutoDelete 92
- TDC member functions
 - AngleArc 93
 - Arc 93
 - BeginPath 93
 - BitBlt 94
 - Chord 94
 - CloseFigure 95
 - DPToLP 95
 - DrawFocusRect 95
 - DrawIcon 95
 - DrawText 95
 - Ellipse 97
 - EndPath 97
 - EnumFontFamilies 97
 - EnumFonts 97
 - EnumMetaFile 98
 - EnumObjects 98
 - ExcludeClipRect 98
 - ExcludeUpdateRgn 99
 - ExtFloodFill 99
 - ExtTextOut 99
 - FillPath 100
 - FillRect 100
 - FillRgn 100
 - FlattenPath 101
 - FloodFill 101
 - FrameRect 101
 - FrameRgn 101
 - GetAspectRatioFilter 101
 - GetAttributeHDC 136
 - GetBkColor 101
 - GetBkMode 101
 - GetBoundsRect 102
 - GetBrushOrg 102
 - GetCharABCWidths 102
 - GetCharWidth 102
 - GetClipBox 103
 - GetClipRgn 103
 - GetCurrentObject 103
 - GetCurrentPosition 103
 - GetDCOrg 103
 - GetDeviceCaps 104
 - GetDIBits 104
 - GetFontData 104
 - GetGlyphOutline 110
 - GetHDC 136

GetKerningPairs 104
 GetMapMode 105
 GetNearestColor 106
 GetOutlineTextMetrics 106
 GetPixel 106
 GetPolyFillMode 106
 GetROP2() 106
 GetStretchBltMode 106
 GetSystemPaletteEntries 107
 GetSystemPaletteUse 107
 GetTabbedTextExtent 107
 GetTextAlign 107
 GetTextCharacterExtra 129
 GetTextColor 109
 GetTextExtent 109
 GetTextFace 109
 GetTextMetrics 110
 GetViewportExt 110
 GetViewportOrg 111
 GetWindowExt 111
 GetWindowOrg 111
 GrayString 111
 Init 136
 IntersectClipRect 112
 InvertRect 112
 InvertRgn 112
 LineDDA 113
 LineTo 113
 LPtoDP 113
 MaskBlt 114
 ModifyWorldTransform 115
 MoveTo 115
 OffsetClipRgn 116
 OffsetViewportOrg 116
 OffsetWindowOrg 116
 operator HDC() 116
 PaintRgn 116
 PatBlt 116
 PathToRegion 117
 Pie 117
 PlayMetaFile 118
 PlayMetaFileRecord 118
 PlgBlt 118
 PolyBezier 119
 PolyBezierTo 119
 PolyDraw 119
 Polygon 120
 Polyline 120
 PolylineTo 121
 PolyPolygon 121
 PolyPolyline 121
 PtVisible 121
 RealizePalette 121
 Rectangle 122
 RectVisible 122
 ResetDC 122
 RestoreBrush 122
 RestoreDC 122
 RestoreFont 122
 RestoreObjects 122
 RestorePalette 123
 RestorePen 123
 RestoreTextBrush 123
 RoundRect 123
 SaveDC 123
 ScaleViewportExt 123
 ScaleWindowExt 123
 ScrollDC 124
 SelectClipPath 124
 SelectClipRgn 125
 SelectObject 125
 SelectStockObject 125
 SetBkColor 125
 SetBkMode 126
 SetBoundsRect 126
 SetBrushOrg 126
 SetDIBits 126
 SetDIBitsToDevice 127
 SetMapMode 127
 SetMapperFlags 127
 SetMiterLimit 128
 SetPixel 128
 SetPolyFillMode 128
 SetROP2 128
 SetStretchBltMode 129
 SetSystemPaletteUse 129
 SetTextAlign 129
 SetTextCharacterExtra 129
 SetTextColor 130
 SetTextJustification 130
 SetViewportExt 130
 SetViewportOrg 130
 SetWindowExt 131
 SetWindowOrg 131

- SetWorldTransform 131
- StretchBlt 131
- StretchDIBits 132
- StrokeAndFillPath 132
- StrokePath 132
- TabbedTextOut 132
- TextOut 133
- TextRect 133
- UpdateColors 134
- WidenPath 134
- TDecoratedFrame
 - inheritance diagram 496
- TDecoratedFrame class 136
 - constructor 136
 - data members, protected 137
 - macro 138
 - member functions
 - protected 137
 - public 136
 - response table 138
- TDecoratedMDIFrame
 - inheritance diagram 497
- TDecoratedMDIFrame class 138
 - constructor 138
 - member functions, protected 138
 - response table 139
- TDesktopDC
 - constructor 139
- tdGetData
 - TRangeValidator enum 338
 - TTransferDirection enum 394
- tdGetData constant 80, 236, 254, 357, 374
- TDialog
 - inheritance diagram 499
- TDialog class 139
 - constructor 140
 - data members, public 140
 - destructor 140
 - member functions
 - protected 143
 - public 140
 - response table 144
- TDialogAttr structure 144
- TDib
 - constructors 146
 - data members
 - Bits 145
 - H 145
 - Info 145
 - IsCore 145
 - Mode 145
 - NumClrs 146
 - W 146
 - destructor 147
 - member functions
 - GetInfo 148
- TDib member functions
 - ChangeModeToPal 147
 - ChangeModeToRGB 147
 - FindColor 147
 - FindIndex 147
 - GetBits 148
 - GetColor 148
 - GetColors 148
 - GetIndex 148
 - GetIndices 148
 - GetInfoHeader 148
 - Height 148
 - InfoFromHandle 151
 - IsOk 148
 - IsPM 149
 - LoadFile 151
 - LoadResource 151
 - MapColor 149
 - MapIndex 149
 - numColors 149
 - numScans 149
 - operator BITMAPINFO() 149
 - operator BITMAPINFOHEADER() 150
 - operator HANDLE() 150
 - operator TRgbQuad() 150
 - Read 151
 - SetColor 150
 - SetIndex 150
 - Size 150
 - StartScan 150
 - ToClipboard 150
 - Usage 150
 - Width 151
 - WriteFile 151
- TDibDC
 - constructor 151, 152
- TDocManager
 - inheritance diagram 500

- TDocManager class constructor *153*
- TDocManager class *152*
 - data members
 - public *153*
 - macro *157*
 - member functions
 - protected *156*
 - public *153*
 - response table *157*
- TDocTemplate
 - inheritance diagram *500*
- TDocTemplate class *157*
 - constructor *160*
 - destructor *160*
 - member functions
 - protected *160*
 - public *157*
 - templates, creating *20*
- TDocTemplateT class *160*
 - member functions
 - public *161*
- TDocument
 - and closing files *192*
 - inheritance diagram *500*
- TDocument::List class *169*
 - constructor *169*
- TDocument class *162*
 - constructor *163*
 - data members
 - protected *168*
 - public *163*
 - destructor *163*
 - member functions
 - protected *168*
 - public *164*
 - templates, creating *20*
- TDropInfo
 - class *169*
 - constructors *169*
- TDropInfo member functions
 - DragFinish *170*
 - DragQueryFile *170*
 - DragQueryFileCount *170*
 - DragQueryFileNameLen *170*
 - DragQueryPoint *170*
 - operator HDROP() *171*
- tdSetData
 - TRangeValidator enum *338*
 - TTransferDirection enum *394*
- tdSetData constant *80, 236, 254, 357, 374*
- tdSizeData
 - TRangeValidator enum *338*
 - TTransferDirection enum *394*
- tdSizeData constant *80, 236, 254, 357, 374*
- tdXXXX transfer function constants
 - TInputDialog::TransferData and *236*
 - TListBox::Transfer and *253*
 - TStatic::Transfer and *374*
 - TWindow::TransferData and *443*
- TEdgeConstraint structure *171*
 - member functions, public *171*
- TEdgeOrSizeConstraint structure *172*
 - member functions, public *172*
- TEdit
 - inheritance diagram *501*
- TEdit class *173*
 - constructors *174*
 - data members
 - protected *179*
 - member functions
 - protected *179*
 - public *174*
 - response table *181*
- TEditFile
 - inheritance diagram *503*
- TEditFile class *182*
 - constructor *182*
 - data members, public *182*
 - destructor *182*
 - member functions
 - protected *184*
 - public *183*
 - response table *185*
- TEditSearch
 - inheritance diagram *505*
- TEditSearch class *185*
 - constructor *185*
 - data members, public *185*
 - member functions, public *186*
 - response table *186*
- TEditView
 - inheritance diagram *506*
- TEditView class *187*

- constructor 187
- data members, protected 188
- destructor 187
- member functions, protected 188
- member functions, public 187
- TEllipse
 - TRegion
 - data member 345
- templates
 - creating 18, 20
- TermInstance
 - TApplication member function 39
- test
 - FirstThat function 423
- TEventHandler class 189
- TEventInfo class 190
 - data members, public 190
- TEventStatus enum 191
- Text
 - TTextGadget member function 382
- text
 - buffers 236
 - changing 373
 - Clipboard and 175
 - copying 180
 - deleting 175
 - editing 174
 - finding 197
 - inserting
 - current position 177
 - length 373
 - modifying 177
 - pasting 178
 - replacing 197, 349
 - retrieving user input 235
 - searching for 178, 185
 - selecting 177, 179
 - static controls 374
 - interface element 373
 - status bar 376, 377
- text gadgets 381
- TextLen
 - TComboBox data member 76
 - TStatic data member 373
 - TTextGadget member function 382
- TextOut
 - TDC member function 133
- TextRect
 - TDC member function 133
- TFileBuf
 - and TFileDocument 192
- TFileDocument
 - inheritance diagram 507
- TFileDocument class 192
 - constructors 192
 - destructor 192
 - member functions 192
- TFileOpenDialog
 - inheritance diagram 508
- TFileOpenDialog class 195
 - constructor 195
 - member functions, public 195
- TFileSaveDialog
 - inheritance diagram 510
- TFileSaveDialog class 195
 - constructor 195
 - member functions, public 196
- TFileStreamBase
 - and TFileDocument 192
- TFilterValidator class 196
 - constructor 196
 - data members, protected 197
 - member functions, public 196
- TFindDialog
 - inheritance diagram 512
- TFindDialog class 197
 - constructor 197
 - member functions, protected 197
- TFindReplaceDialog::TData struct
 - member functions
 - public 199
- TFindReplaceDialog::TData structure 199
 - errors 199
- TFindReplaceDialog class 197
 - constructor 198
 - data members, protected 198
 - flags 200
 - member functions
 - protected 199
 - public 198
- TFindReplaceDialog structure 199
- TFloatingFrame
 - inheritance diagram 514
- TFloatingFrame class 384

- data members
 - public 201
- macros 202
- response table 202
- TFont
 - constructors 202
 - data members
 - Stocks[] 202
 - TStockId 202
 - member functions
 - GetObject 203
 - GetStock 203
 - HFONT() 203
- TFrameWindow class 203
 - constructors 204
 - data members
 - protected 206
 - public 204
 - destructor 204
 - member functions
 - protected 206
 - public 204
 - response table 208
- TGadgetWindow
 - inheritance diagram 516
 - response table 221
- TGadgetWindowFont class 221
- TGadgetWithId
 - TGadgetWindow member function 216
- TGauge
 - inheritance diagram 518
- TGdiObject 224
- TGdiObject constructors 227
- TGdiObject data members
 - enum TAutoDelete 225
 - enum TType 225
 - Handle 227
 - ShouldDelete 227
- TGdiObject destructor 228
- TGdiObject macros
 - OBJ_REF_ADD 228
 - OBJ_REF_COUNT 228
 - OBJ_REF_DEC 229
 - OBJ_REF_INC 229
 - OBJ_REF_REMOVE 229
- TGdiObject member functions
 - GetObject 225
 - IsGDIObject 226
 - IsOK 226
 - RefAdd 226
 - RefCount 226
 - RefDec 226
 - RefFind 226
 - RefInc 227
 - RefRemove 227
- TGroupBox class 230
 - constructors 231
 - data members, public 231
 - member functions
 - protected 232
 - public 231
- TheClipboard
 - TClipboard data member 70
- THintMode enum 232
- thread processing 430
- Throw
 - TXInvalidMainWindow public member function 40
 - TXOwl public member function 455
- throwing exception objects 40
- thumb, scroll bars 354, 355, 360, 361
 - moving 356, 357
 - returning position 358
- ThumbRect
 - TSlider data member 372
- ThumbResId
 - TSlider data member 372
- ThumbRgn
 - TSlider data member 372
- TIC
 - constructors 233
- TicGap
 - TSlider data member 372
- tick positions
 - sliders and 366, 368
- TIcon 233
 - constructors 234
 - destructor 234
 - member functions
 - GetIconInfo 235
 - HICON() 235
- TileChildren
 - TMDIClient member function 265

- TileGadgets
 - TGadgetWindow member function 221
 - TToolBox member function 391
- tiling 265
 - direction 390
- timer event 433, 441
- TInputDialog class 235
 - constructor 236
 - control IDs 22
 - data members, public 235
 - member functions
 - protected 236
 - public 236
- TInStream class 236
- Title
 - retrieving 431
 - TPrintout data member 332
 - TWindow data member 414
- TKeyboardModeTracker class
 - macros 239
 - response table 239
- TLayoutConstraint structure 239
 - constants 241
 - data members, public 240
 - TWidthHeight enum and 412
- TLayoutMetrics class 241
 - constructor 242
 - data members, public 242
- TLayoutWindow
 - inheritance diagram 519
- TLayoutWindow class 244
 - constructor 247
 - data members, protected 248
 - destructor 247
 - member functions
 - protected 248
 - public 247
 - response table 248
- TListBox
 - inheritance diagram 520
- TListBox class 248
 - constructors 248, 249
 - member functions
 - protected 254
 - public 249
- TListBoxData structure 254
 - constructor 255
 - data members, public 254
 - destructor 255
 - member functions, public 255
- TListView
 - inheritance diagram 521
- TListView class
 - constructor 256
 - destructor 256
 - protected data members 257
 - protected member functions 257
 - public data members 256
 - public member functions 256
- TLookupValidator class 260
 - constructor 260
 - member functions, public 260
- TMargins
 - TGadget data member 209
- TMDI frame
 - keyboard navigation 267
- TMDIChild class 261
 - constructors 261
 - destructor 261
 - member functions
 - protected 262
 - public 261
 - response table 263
- TMDIClient
 - inheritance diagram 522
- TMDIClient class 263
 - constructor 263
 - data members, public 263
 - destructor 263
 - member functions
 - protected 265
 - public 263
 - response table 266
- TMDIFrame
 - inheritance diagram 523
- TMDIFrame class 267
 - constructors 267
 - data members, public 267
 - member functions
 - protected 268
 - public 268
 - response table 268
- TMeasurementUnits enum 269

- TMemoryDC
 - constructor 269
- TMemoryDC member functions
 - RestoreBitmap 269
 - RestoreObjects 269
 - SelectObject 270
- TMenu class 270
 - constructors 270, 271
 - data members, public 273
 - destructor 271
 - member functions, public 271
- TMenuDescr::TGroup enum 276
- TMenuDescr class 273
- TMetaFileDC
 - constructor 277
 - destructor 278
- TMetaFileDC member functions
 - Close 278
- TMetaFilePict
 - class 278
 - constructors 278
 - destructor 279

- TMetaFilePict member functions
 - GetMetaFileBits 280
 - GetMetaFileBitsEx 280
 - IsOK 280
 - operator HMETAFILE() 280
- TModeIndicator
 - TKeyboardModeTracker data member 237
- TModule class 280
 - constructors 281
 - data members
 - protected 287
 - public 280
 - destructor 281
 - member functions, public 282
- ToClipboard
 - TBitmap member function 43
 - TDib member function 150
 - TPalette member function 296
- Toggle
 - TCheckBox member function 60
- ToggleModeIndicator
 - TStatusBar member function 377

- ToGroupEnd
 - TPXPictureValidator member function 336
- toolbars 136
- toolbox gadgets 389
- top window 429
- ToPage
 - TPrintDialog::TData struct data member 322
- TOpenSaveDialog::TData struct
 - data members
 - public 290
- TOpenSaveDialog class 287
 - constructors
 - protected 288
 - public 288
 - data members, protected 288
 - member functions
 - protected 289
 - public 288
 - response table 290
- TopLeft
 - TRect member function 344
- TopRight
 - TRect member function 344
- Touches
 - TRect member function 344
 - TRegion member function 348
- TOutStream class 292
- TPaintDC
 - constructor 292
 - destructor 293
- TPaintDC data members
 - OrgBitmap 270
 - Ps 292
- TPalette 293
 - constructors 293
 - data members
 - Stocks[] 293
 - TStockId 293
 - member functions
 - AnimatePalette 295
 - Create 297
 - GetNearestPaletteIndex 295
 - GetNumEntries 295
 - GetObject 295
 - GetPaletteEntries 295
 - GetPaletteEntry 295
 - GetStock 295

- HPALETTE() 296
- ResizePalette 296
- SetPaletteEntries 296
- SetPaletteEntry 296
- ToClipboard 296
- UnrealizeObject 297
- related operator
 - << 296
- TPaletteEntry
 - class 297
 - constructors 298
- TPen 298
 - constructors 299
 - data members
 - enum TStockId 298
 - Stocks[] 299
 - member functions
 - GetObject 300
 - GetStock 300
 - operator HPEN() 300
- TPicResult enum 300
- TPlacement enum 301
- TPoint
 - class 301
 - constructors 301
- TPoint friends
 - operator<< 303
 - operator>> 303
- TPoint member functions
 - Offset 302
 - OffsetBy 302
 - operator+ 302
 - operator- 302
 - operator!= 303
 - operator+= 303
 - operator-= 303
 - operator== 303
- TPointer
 - class 304
- TPointer class
 - constructor 304
 - member functions
 - public 304
- TPopupMenu class 305
 - constructor 305
 - memberfunctions 305
- TPreviewPage class 306
 - constructor 307
 - data members, protected 307
 - member functions
 - protected 308
 - public 307
 - response tables 308
- TPrintDC
 - constructor 308
- TPrintDC associated functions
 - DeviceCapabilities 309
- TPrintDC data members
 - DocInfo 319
- TPrintDC member functions
 - AbortDoc 308
 - BandInfo 309
 - EndDoc 312
 - EndPage 312
 - Escape 313
 - NextBand 317
 - QueryAbort 317
 - QueryEscSupport 317
 - SetAbortProc 317
 - SetCopyCount 318
 - StartDoc 318
 - StartPage 318
- TPrintDialog::TData struct
 - data members, public 321
 - flags 321
- TPrintDialog class 319
 - constructor 319
 - data members, protected 320
 - error codes 321
 - member functions, public 319
 - response table 320
 - TData structure 320
- TPrinter class 327
 - constructor 327
 - data members, protected 328
 - destructor 327
 - error codes
 - returning 328
 - member functions
 - protected 329
 - public 328
- TPrinterAbortDlg class 329
 - constructor 329
 - member functions, public 330

- TPrintout class 330
 - constructor 330
 - data members, protected 332
 - destructor 330
 - member functions, public 330
- TPrintPreviewDC class 323
 - constructor 324
 - data members, protected 327
 - member functions
 - protected 327
 - public 324
- TProcInstance class 333
 - constructor 333
- TPXPictureValidator class 333
 - constants 334
 - constructor 333
 - data members, protected 335
 - member functions
 - protected 335
 - public 334
- tracking keyboard modes 237
- TrackMenuSelection
 - TDecoratedFrame data member 137
- TrackMode
 - TScroller data member 359
- TrackMouse
 - TGadget data member 212
- TrackPopupMenu
 - TPopupMenu member function 305, 306
- TRadioButton
 - inheritance diagram 525
- TRadioButton class 336
 - constructor 336
 - member functions, protected 337
 - response table 337
- TRangeValidator class 337
 - constructor 337
 - data members, protected 338
 - member functions, public 337
- Transfer
 - TCheckBox member function 60
 - TComboBox member function 80
 - TListBox member function 253
 - TRangeValidator member function 338
 - TScrollBar member function 357
 - TStatic member function 374
 - TValidator member function 393
 - TWindow member function 443
- transfer buffers 446
- transfer functions 391
- transfer mechanism
 - buffers 446
 - interface objects
 - disabling 421
 - enabling 422
 - tdxxx constants 446
- TransferBuffer
 - TWindow data member 445
- TransferData
 - TInputDialog member function 236
 - TWindow member function 443
- TransferDC
 - TPrintDialog::TData member function 323
- TRect
 - class 338
 - constructors 339
- TRect friends
 - operator<< 344, 345
 - operator>> 344
- TRect member functions
 - Area 340
 - BottomLeft 340
 - BottomRight 340
 - Contains 340
 - Height 340
 - InflatedBy 341
 - IsEmpty 341
 - IsNull 341
 - Normalize 341
 - Normalized 341
 - Offset 342
 - OffsetBy 342
 - operator& 342
 - operator+ 342
 - operator- 342
 - operator! 342
 - operator!= 343
 - operator&= 343
 - operator+= 343
 - operator-= 343
 - operator== 343
 - operator|= 343
 - operator TPoint*() 343
 - Set 344

- SetEmpty 344
- Size 344
- TopLeft 344
- TopRight 344
- Touches 344
- Width 344
- TRegion 345
 - constructors 345
 - data members
 - enum TEllipse 345
 - member functions
 - Contains 346
 - GetRgnBox 347
 - HRGN() 348
 - operator = 347
 - operator == 347
 - operator != 347
 - operator &= 348
 - operator += 347
 - operator -= 347
 - operator ^= 348
 - operator |= 348
 - SetRectRgn 348
 - Touches 348
- TRelationshipUnits enum 349
- TReplaceDialog class 349
 - constructors 349
 - member functions, protected 349
- TResID
 - class 350
 - constructors 350
- TResId
 - TMenuDescr data member 275
- TResID friends
 - operator << 350, 351
 - operator >> 350
- TResID member functions
 - IsString 350
 - operator LPSTR() 350
- TResponseTableEntry class 351
 - data members, public 351
- TRgbQuad
 - class 352
 - constructors 352
- TRgbTriple
 - class 353
 - constructors 353
- try keyword
 - and exceptions 453
- TScreenDC
 - constructors 354
- TScrollBar class 354
 - constructors 355
 - constructors 355
 - data members, public 354
 - member functions
 - protected 357
 - public 355
 - warning 354
- TScrollBarData structure 358
 - data members, public 358
- TScroller
 - conversions
 - range values 26
 - scroll values 26
- TScroller class 358
 - constructor 359
 - destructor 360
 - member functions, public 360
 - public data members 359
- TSeparatorGadget class 362
- TSize
 - class 363
 - constructors 363
- TSize friends
 - operator << 365
 - operator >> 365
- TSize member functions
 - Magnitude 364
 - operator+ 364
 - operator- 364
 - operator!= 364
 - operator+= 364
 - operator-= 365
 - operator== 364
- TSlider
 - inheritance diagram 527
- TSlider class 365
 - constructors 367
 - data members, protected 371
 - destructors 367
 - member functions
 - protected 368
 - public 367

- response table 372
- TSockId
 - TPen data member 298
- TSortedStringArray typedef 373
- TState
 - TButtonGadget data member 51
 - TButtonGadget member function 51
- TStatic class 373
 - constructors 373, 374
 - data members, public 373
 - member functions
 - protected 375
 - public 374
- TStatus
 - TStatus data member 375
- TStatus class
 - data members
 - public 375
- TStatusBar
 - inheritance diagram 529
 - TStatusBar data member 376
- TStatusBar class 375
 - constructor 376
 - data members
 - protected 377
 - public 376
 - member functions
 - protected 378
 - public 376
- TStockId
 - TFont data member 202
 - TPalette data member 293
- TStream class 378
 - public data members 378
 - public destructor 378
 - public member functions 379
- TStringLookupValidator class 379
 - constructor 379
 - data members, protected 380
 - destructor 379
 - member functions, public 380
- TSystemMenu class 380
 - constructor 380
- TTextGadget
 - inheritance diagram 530
- TTileDirection enum 383
- TTinyCaption class 383
 - constructors 385
 - data members, protected 384
 - destructors 385
 - macros 389
 - member functions, protected 385
 - response table 389
- TToolBox
 - inheritance diagram 531
- TTransferDirection enum 391
- TType
 - TButtonGadget data member 51
 - TGdiObject data member 225
- TValidator class 391
 - constructor 392
 - data members, protected 394
 - destructor 392
 - member functions, public 392
- TVbxControl class 395
 - constructor 396
 - destructor 397
 - member functions
 - protected 401
 - public 397
 - response tables 402
- TVbxEventHandler class 402
 - member functions, protected 407
 - response tables 407
- TView class 407
 - constructor 408
 - data members
 - protected 410
 - public 408
 - destructor 408
 - member functions
 - protected 410
 - member functions, public 409
- TWidthHeight enum 412
- TWindow
 - inheritance diagram 534
- TWindow class 412
 - constructors 414
 - data members
 - protected 445
 - public 413
 - destructors 415
 - flag constants 417, 449

- member functions
 - protected 446
 - public 415
- member functions, defining 83, 84
- response table 447
- TWindowAttr
 - TMDIClient class 263
- TWindowAttr structure 413, 449
 - data members, public 449
 - extended style constants 449
 - style constants 450
 - Tmenu resource ID 450
- TWindowDC
 - constructors 450, 451
 - destructors 450
- TWindowDC data members
 - Wnd 451
- TWindowFlag enum 449
- TWindowView
 - inheritance diagram 533
- TWindowView class 451
 - constructors 451
 - destructors 451
 - member functions, public 451
- twips to pixels conversion 407
- TXCompatibility exception 281
 - and TStatus 375
- TXInvalidMainWindow
 - exception 38
- TXInvalidMainWindow constructor 39
- TXInvalidModule
 - exception 284
- TXInvalidWindow
 - and TEditView 256
- typographical conventions 2

U

- UINT
 - TMenu member function 272
- Uncheck
 - TCheckBox member function 60
- Undo, TEdit member function 179
- Unhandled
 - TXOwl public member function 455
- Units
 - TLayoutConstraint data member 241

- Unlock
 - TPrintDialog::TData member function 323
- UnlockBuffer
 - TEdit member function 179
- UnrealizeObject
 - TBrush member function 48
 - TPalette member function 297
- UnRefTemplate
 - TDocManager member function 156
- UnregisterHotKey
 - TWindow member function 444
- UnsetOption
 - TValidator member function 394
- Update
 - TControlGadget member function 89
- update rectangle
 - windows 429
- update region
 - windows 430
- UpdateColors
 - TDC member function 134
- UpdateStatusBar
 - TKeyboardModeTracker data member 238
- UpdateWindow
 - TWindow member function 444
- Usage
 - TDib member function 150
- user input
 - checking 196, 260
 - data entry 391
 - input fields 379
 - numeric values 337
 - picture strings 333
- user input, retrieving 235
- user interface
 - bitmaps 55

V

- Valid
 - TValidator member function 394
- valid characters
 - input fields 197
 - picture formats 335
- Validate
 - TWindow member function 444
- VALIDATE.CPP 391

- ValidateRect
 - TWindow member function 444
- ValidateRgn
 - TWindow member function 444
- validating
 - picture result type 300
 - pictures 300
- validating edits
 - and derived classes 179
- validating user input 196, 260
 - data entry 391
 - input fields 379
 - numeric values 337
 - picture strings 333
- Validator
 - TEdit data member 179
- validator constants 455
- validators
 - data transfer 338, 393
 - filter 196
 - IDs
 - constants 25
 - lookup 260
 - string 379
 - picture 333
 - range 337
 - validity testing 197, 380
 - picture formats 334
- ValidChars
 - TFilterValidator data member 197
- validity testing 393, 394
- ValidWindow
 - TModule member function 287
- Value
 - TColor data member 75
 - TGauge data member 223
- Value constant 241
- values
 - checking 393
 - range of 337
 - setting 197
 - maximum/minimum 338
- VBX controls
 - firing events 403
 - receiving events 402
- VBX_EVENTARGNUM 406
- VBXEVENT structure
 - Control 404
 - EventIndex 404
 - EventName 404
 - ID 404
 - lparam 404
 - NumParams 404
 - ParamList 404
 - Window 404
- version numbers
 - returning 30
- vertical scroll bars 354
- vertical scrollbar range value
 - converting 362
- vertical sliders 411
- viewing
 - current state, program 376, 377
- viewport 324, 325, 326
- views
 - closing 20
 - creating 20
 - ID constants 25
 - with no window 408
- virtual key codes 237
- VnCommit
 - TEditView member function 188
 - TListView member function 259
- VnDocClosed
 - TEditView member function 188
 - TListView member function 259
- VnIsDirty
 - TEditView member function 188
 - TListView member function 259
- VnIsWindow
 - TEditView member function 189
 - TListView member function 259
- VnRevert
 - TEditView member function 189
 - TListView member function 259
- Vnxxxx view notification constants 455
- Voxxxx validator constants 455
- VScroll
 - TScroller member function 361

W

- W
 - TDib member function 146

- TWindowAttr structure 450
- WaitingForSysCmd
 - TTinyCaption data member 384
- warning beeps 180
- wfAutoCreate constant 417
- wfXxxx constants 417
- White
 - TColor data member 73
- WideAsPossible
 - TGadget data member 209
 - TGadgetWindow data member 219
- WidenPath
 - TDC member function 134
- Width
 - TBitmap member function 43
 - TDib member function 151
 - TLayoutMetrics data member 242
 - TRect member function 344
- Win32
 - icons 2
- WIN32 DLLs
 - exporting 29
 - importing 29
- WIN API
 - encapsulated functions 473
- Window
 - TGadget data member 213
 - TScroller data member 359
 - TXWindow data member 448
- window classes
 - returning information on 283
- window mapping functions 324, 325, 326
- windowev.h
 - window messages 457
- WindowFromPoint
 - TWindow member function 444
- WindowGroup
 - TMenuDescr enum 276
- WindowProc
 - TWindow member function 444
- Windows
 - bitmaps, predefined 284
 - combo box
 - interface element 76
 - types 76
 - control interface elements 84
 - control message 143
 - CreateDialogParam function 141
 - CTL3D DLL support 139
 - cursors, predefined 285
 - default
 - class name, modal dialog box 143
 - message processing 413
 - DialogBoxParam function 142
 - edit control interface element 173
 - EndDialog function 141
 - handles, returning 286
 - icons, predefined 285
 - interface element 417
 - list box interface element 248
 - OPENFILENAME structure 195
 - predefined class 207
 - push button interface element 48
 - radio button interface element 336
 - redisplay 434
 - registration class attributes 446
 - registration class name
 - "BUTTON" 50, 60, 232
 - "EDIT" 181
 - "LISTBOX" 254
 - "MDICLIENT" 266
 - "STATIC" 375
 - class attributes and 435
 - pure virtual function 446
 - resources, loading into memory 286
 - SB_ constants 360
 - scroll bar messages 354
 - static text interface element 373
- windows
 - bit mask constants 449
 - caption bars, creating 383
 - cascading 264
 - child 138, 141, 239, 241, 261, 263, 264
 - client 136, 203
 - Clipboard-viewer chain
 - adding 70, 71
 - removing 71
 - closing 34, 264, 417
 - dialog box 140, 142
 - constraints
 - defined 241
 - edge 171, 241, 244
 - layout 239, 241
 - creating 141, 239, 412

- main 31, 38
- decorating 136, 138
- default procedure 413
- default processing 139
- handles
 - retrieving 68
- layout 349
 - constraints 239, 241
 - metrics, defining 241, 244
- main 267, 417
 - status 281
- moving through 203, 358
- naming 39, 412, 414
- painting 207
- placing 171
- property list, retrieving handle for 428
- sizing 269, 349, 412
- tiling 265
- WinHelp
 - TWindow member function 445
- WM_CREATE message 450
- WM_INITDIALOG message
 - TDialogAttr and 145
- WM_PAINT message 207, 434
- WM_TIMER messages 433, 441
- WM_VBXFIREEVENT message 403
- Wnd
 - TWindowDC data member 451
- word-break functions 179
- wordwrapping 175, 177
- WParam
 - control message parameter 143
- Write
 - TEditFile member function 184
- WriteFile
 - TDib member function 151
- WS_ window style constants 76, 84, 336, 374
- WS_BORDER constant 248
- WS_HSCROLL constant 354
- WS_TABSTOP style 231
- WS_VSCROLL constant 248, 354
- WS_XXX window style constants 263

X

- X
 - TLayoutMetrics data member 242
 - TWindowAttr structure 450
- XLine
 - TScroller data member 359
- xmsg error
 - TModule function 282
- XPage
 - TScroller data member 359
- XPos
 - TScroller data member 359
- XRange
 - TScroller data member 359
- XRangeValue
 - TScroller member function 361
- xs exception status bit flags 37
- xs exception status enum 456
- XScrollValue
 - TScroller member function 361
- XUnit
 - TScroller data member 359

Y

- Y
 - TLayoutMetrics data member 242
 - TWindowAttr structure 450
- YLine
 - TScroller data member 359
- YPage
 - TScroller data member 359
- YPos
 - TScroller data member 359
- YRange
 - TScroller data member 359
- YRangeValue
 - TScroller member function 362
- YScrollValue
 - TScroller member function 362
- YUnit
 - TScroller data member 359



Borland

Corporate Headquarters: 100 Borland Way, Scotts Valley, CA 95066-3249, (408) 431-1000. Offices in: Australia, Belgium, Canada, Denmark, France, Germany, Hong Kong, Italy, Japan, Korea, Latin America, Malaysia, Netherlands, New Zealand, Singapore, Spain, Sweden, Taiwan, and United Kingdom • Part # BCP1240WW21773 • BOR 6273