



User's Guide for IWBasic

© 2010 - 2016 Ionic Wind Software All Rights Reserved.

IWBasic

User's Guide

by Ionic Wind Software

IWBASIC User's Guide - WIP

© 2010 - 2016 Ionic Wind Software All Rights Reserved.

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: August 2016 in USA.

Publisher

IonicWind Software

Technical Editors

T. L. McCaughn

Cover Designer

IonicWind Software

Special thanks to:

To all the people who have helped make IWBasic a better product.

Table of Contents

Part I Introduction	40
Part II Getting Started: Beginners	44
Part III Getting Started: Upgrading	48
Part IV IDE Interface	50
1 Overview.....	50
2 Caption Bar.....	50
3 Main Menu.....	51
4 Main ToolBar.....	64
5 WorkSpace.....	66
6 Project List Window.....	67
Introduction	67
Files	68
Subroutines	68
Includes	69
7 Output Window.....	71
Introduction	71
Build	72
Debug	72
Find In Files	73
Resources	73
Part V IDE Modes	78
1 Single file programs.....	78
Introduction	78
Steps to use single file mode	78
Executing your program	78
Notes	78
2 Multi-file programs.....	79
Introduction	79
Steps to use multi file mode	79
Executing your program	80
The \$main directive	80
Part VI Utilities	82
1 Code Editor.....	82
Introduction	82
Work Area	82
Title Bar	86
Toolbar	87

Bookmarks.....	87
Code Folding.....	88
Finding Text	89
Regular Expression Help	90
Replacing Text	92
Commenting	93
AutoComplete	95
AutoTip	95
Editing Text	96
Saving the File	96
File encoding	97
2 Form Editor.....	97
Introduction	97
Creating a new Form	101
Sizing the Form	101
Form Properties	101
Adding Controls	105
Moving and Sizing controls	106
Changing control properties	107
Testing the Form	107
Generating source code	108
Saving a Form	110
3 Menu Editor.....	111
Introduction	111
Getting Started	115
Editing	115
Deleting / Moving Entries	117
Preview	117
4 Create Import Library.....	119
5 Register File Extensions.....	120
6 Tools Menu Editor.....	121
Introduction	121
Adding Tools	121
Deleting Tools	122
7 Help Menu Editor.....	124
Introduction	124
Adding Help Files	125
Deleting Help Files	126
8 File Cleanup.....	128
9 ColorPicker.....	130
10 Toolbar Paint.....	132
11 MessageBox Builder.....	133
12 Print Preview.....	137
Introduction	137
Caption	137
Main Toolbar	137
Printer	138
Orientation	139
Print Range	139
Copies	140

Page Order	140
Margins	141
Options	141
Preview Window	142
13 Find in Files.....	146
Part VII To-Do Lists	152
1 Master ToDo List.....	152
2 Project ToDo List.....	153
Part VIII Language	156
1 Language Syntax Overview.....	156
Source Files	156
Identifiers	156
Reserved Words	156
Commands, Statements and Functions	157
Resolving syntax conflicts	157
Source comments	158
2 Constants and Literals.....	158
String literals	159
Unicode literals	160
"T" string literals	161
Numeric literals	161
Numeric modifiers	162
Overriding defaults	163
3 Variables.....	163
Source global variables	163
Program global variables	163
Local variables	164
Defining variables	164
Variable types	164
Variable Assignments	165
Post assignment	165
Global Initializers	165
Arrays	166
The ISTRING type	167
The IWSTRING type	167
The TSTRING type	167
The ITSTRING type	167
The OEM type	163
User defined variable types (UDT)	168
Unions	169
Creating aliases for variable types with TYPEDEF	169
4 Operators.....	171
Mathematic operators	172
Compound operators and prefix/postfix operators	172
Conditional operators	173
Boolean operators	174
Control operators	174
STRING operators	175
String evaluation	175

String concatenation operator	175
String multiplication operator	176
Operator Precedence	176
Type Promotion	177
5 Pointers and Typcasting.....	177
Type casting	179
Pointer math and array indexes	180
Multiple indirection	180
6 Conditional Statements.....	181
IF Statement	181
ELSEIF statement	183
SELECT Statement	183
In-Line IIF Statement	185
7 Loop Statements.....	185
FOR Statement	185
DO and WHILE statements	186
FOR EACH statement	187
8 Subroutines / Functions.....	187
Calling the subroutine	188
Passing parameters	189
Passing Arrays	189
Optional Parameters	190
Declaring subroutines	190
Global Subroutines	191
Indirectly calling subroutines	191
Subroutines with a variable number of arguments	192
Labels	187
Special return values	187
9 Using Linked Lists.....	194
Creating a new list	195
Adding data to the list	195
Iterating a list	195
Removing elements while iterating	196
Removing all elements and deleting the list	196
10 Using DATA Statements.....	197
Defining a block of data	197
Reading data	197
Restoring the data pointer	198
11 Using DLL's and the Windows API.....	198
Creating import libraries	198
Including the import library in the build	199
Calling functions in the DLL	199
Calling functions in the C runtime library	199
Calling functions with C name mangling	200
Importing variables	200
Import libraries included with IW BASIC as of Version 2.0	200
12 Conditional Compiling.....	201
Creating a conditional identifier	201
Excluding or including code to be compiled	202
Notes on conditional compiling	203
13 \$INCLUDE Command.....	203

Notes, caveats, warnings	204
14 Using COM.....	205
The interface	205
Creating the object	206
Calling methods in the object	207
Interfaces within interfaces	207
GUID usage	208
15 Object Oriented Programming.....	208
Class Definitions	208
Creating and using the Object	209
Constructors and Destructors	210
Access protection	212
Inheritance	213
Method overriding and polymorphism	214
Scope Resolution	215
The THIS pointer	216
Imported methods	216
Design considerations	217
16 Inline Assembly.....	217
Referring to global variables	217
Referring to local variables	218
Assembly Data Segment	219
Labels in assembly code	219
Register usage and context	220
17 Compiler Options.....	220
Global Options	221
\$Option keyword	221
/a	227
/m	227
AsmVariables.....	223
CaseSensitive.....	225
Codepage.....	222
DimOrder.....	228
Double	222
ErrorLimit.....	222
Float	222
IncludePath.....	223
Intrinsic	224
LibPath	227
NoVtable.....	225
OldLibsLocation.....	226
Optimization.....	224
QuoteLibPaths	227
ResolveLibs.....	228
Return	224
Strict	226
Warning	223
ZeroVariables.....	226
Command line	228
18 Exception Handling.....	230
Exception Codes	237
19 Threads.....	239

20	Macros.....	239
Part IX General Programming		244
1	Text Only Programs.....	244
	Console Window	244
	Displaying Text	244
	Changing Colors	245
	Positioning Text	245
	Clearing The Window	246
	Waiting for a key	246
	Getting Input	247
2	File Operations.....	248
	Opening files for reading and writing	248
	Closing open files	248
	Writing to the file	249
	Reading from the file	250
	Determining the length of files	251
	Moving the file pointer	251
	Random Access Files	251
	Copying files	252
	Deleting files	252
	Creating Directories	253
	Removing directories	253
	Reading directories	253
	Opening the system file dialog	254
3	Formatting Output.....	255
	Truncation and rounding	256
	Filling with spaces or 0	256
	Justification	256
	Including string variables	257
	Expected types and overriding defaults	257
4	Using Strings.....	257
	Common String Functions	258
	APPEND\$.....	258
	ASC	258
	CHR\$/WCHR\$.....	258
	DATES\$	259
	DATES\$(format).....	259
	HEX\$/WHEX\$.....	259
	INSTR/WINSTR.....	260
	LCASE\$/WLCASE\$.....	260
	LEFT\$/WLEFT\$.....	260
	LEN	260
	LTRIM\$/WLTRIM\$.....	260
	MID\$/WMID\$.....	261
	REPLACE\$.....	262
	RIGHT\$/WRIGHT\$.....	261
	RTRIM\$/WRTRIM\$.....	261
	SPACE\$/WSPACE\$.....	261
	STR\$/WSTR\$.....	261
	STRING\$/WSTRING\$.....	262
	TIME\$	262

UCASE\$/WUCASE\$.....	262
VAL/WVAL.....	262
Converting between Unicode and ANSI	263
5 Writing DLL's.....	263
The EXPORT statement	263
Notes	264
6 Using resources.....	264
7 MIDI Music and Sound.....	266
Playing wave files	266
Playing MIDI streams	267

Part X Windows Programming

270

1 Creating a Window.....	270
Opening a Window	270
System variables and constants	271
Creation style flags	271
Waiting for messages	272
Handling Messages	272
Returning Values	273
2 Messages and Message Loops.....	273
Introduction	273
The message queue and loop	273
The WAITUNTIL statement.....	273
The WAIT statement	274
Message ID's and the handler	275
Mouse messages.....	275
Keyboard messages.....	277
Window and system messages.....	277
3 Printing Text in a Window.....	279
Changing the font	280
Selecting character sets	280
Changing text colors	281
FONTREQUEST function	281
4 Graphics and Drawing.....	282
LINE and LINETO statements	282
RECT statement	282
ELLIPSE statement	282
CIRCLE statement	283
PSET statement	283
GETPIXEL function	283
Drawing Modes	283
Raster operations	284
Advanced graphics statements	284
The @NOAUTODRAW flag	284
5 Images, Icons and Cursors.....	285
Loading an image	285
Freeing the image	286
Displaying the image	286
Changing cursors	286
Changing icons	287

6	Creating and Using Menus.....	287
	Adding a menu bar to a window or dialog	287
	Creating submenus (popup menus)	288
	Inserting menus	288
	Creating context menus	288
	Item addition and removal	289
	Menu appearance	289
	Handling menu messages	289
	Keyboard Accelerators	290
	Low level API menu functions	291
7	Creating Embedded Browsers.....	291
	Controlling the browser	292
	Navigating	293
	Messages	294
	Notes	295
8	Using Dialogs.....	295
	Creating a Dialog	295
	Showing the dialog	295
	Closing the dialog	296
9	MDI Windows.....	297
10	Information Functions.....	299
	Retrieving sizes	299
	GETSIZE.....	299
	GETCLIENTSIZE.....	299
	GETSCREENSIZE.....	299
	GETTEXTSIZE.....	299
	Retrieving drawing and caret positions	299
	GETPOSITION.....	299
	GETCARETPOSITION.....	299
	Miscellaneous	299
	GETCAPTION.....	299
	GETCONTROLTEXT.....	299
	GETHDC	299
11	ScrollBars.....	300
	SETSCROLLRANGE	300
	GETSCROLLRANGE.....	300
	SETSCROLLPOS	300
	GETSCROLLPOS	300
	GETTHUMBPOS	300
12	Controls.....	300
	Control Creation	300
	Introduction.....	300
	Creating other types of controls.....	301
	Windows common controls.....	303
	General Control Functions - WIP	303
	SETCONTROLCOLOR.....	303
	SETCONTROLTEXT.....	303
	GETCONTROLTEXT.....	303
	CONTROLEXISTS.....	303
	SETSCROLLRANGE.....	303
	GETSCROLLRANGE.....	303
	SETSCROLLPOS.....	303

GETSCROLLPOS.....	303
GETTHUMBPOS.....	303
SETSELECTED.....	303
GETSELECTED.....	303
ISSELECTED.....	303
GETSTRINGCOUNT.....	303
GETSTRING.....	303
ADDSTRING.....	303
INSERTSTRING.....	303
DELETESTRING.....	303
SETLBCOLWIDTH.....	303
SETHORIZEXTENT.....	303
SETSTATE.....	303
ENABLECONTROL.....	303
GETSTATE.....	303
SETFOCUS.....	303
SENDMESSAGE.....	303
CONTROLCMD.....	303
Button Controls	304
@Button Controls.....	304
About Button	304
Creating the control.....	304
Button control styles.....	305
Button control functions and statements.....	306
Notification messages.....	307
@RgnButton Controls.....	307
About RgnButton.....	307
Creating the control.....	308
RgnButton control styles.....	308
RgnButton control functions and statements.....	309
Notification messages.....	312
@SysButton Controls.....	312
About SysButton.....	312
Creating the control.....	312
SysButton control styles.....	313
SysButton control functions and statements.....	314
Notification messages.....	315
Calendar Controls	315
About Calendar.....	315
Creating the control.....	318
Calendar control styles.....	318
Calendar control functions and statements.....	320
Notification messages.....	324
Checkbox Controls	324
About Checkbox.....	324
Creating the control.....	324
Checkbox control styles.....	325
Checkbox control functions and statements.....	326
Notification messages.....	327
Combobox Controls	327
About combo box.....	327
Creating the control.....	327
Combo box control styles.....	328
Combo box control functions and statements.....	329

Notification messages.....	331
ComboBoxEx Controls - WIP	332
About combo boxex.....	332
Creating the control.....	332
Combo boxex control styles.....	332
Combo boxex control functions and statements.....	333
Notification messages.....	335
Date/TimePicker Controls - WIP	335
About Date/TimePicker.....	335
Creating the control.....	335
Date/TimePicker control styles.....	335
Date/TimePicker control functions and statements.....	336
Notification messages.....	338
Edit Controls - WIP	338
About edit controls.....	338
Creating the control.....	338
Edit control styles.....	338
Edit control functions and statements.....	340
Clipboard operations.....	341
Line operations.....	342
Selection operations.....	342
Editing operations.....	343
General operations.....	343
Notification messages.....	343
Group Controls - WIP	344
About Group controls.....	344
Creating the control.....	344
Group control styles.....	344
Group control functions and statements.....	344
Notification messages.....	344
Header Controls - WIP	346
About Header.....	346
Creating the control.....	346
Header control styles.....	346
Header control functions and statements.....	347
Notification messages.....	349
IP Controls - WIP	350
About IP.....	350
Creating the control.....	350
IP control styles.....	350
IP control functions and statements.....	350
Notification messages.....	351
ListBox Controls - WIP	352
About list box.....	352
Creating the control.....	352
List box control styles.....	352
List box control functions and statements.....	354
Notification messages.....	356
ListView Controls - WIP	356
About list view.....	356
Creating the control.....	356
List view control styles.....	357
Controlling the list view control.....	359
Statements and Functions.....	359

Notification messages.....	361
Data types.....	362
Reading data types.....	363
Pager Controls - WIP	364
About Pager.....	364
Creating the control.....	364
Pager control styles.....	364
Pager control functions and statements.....	364
Notification messages.....	366
Progress Controls - WIP	366
About Progress.....	366
Creating the control.....	366
Progress control styles.....	367
Progress control functions and statements.....	367
Notification messages.....	369
RadioButton Controls - WIP	369
About RadioButton.....	369
Creating the control.....	369
RadioButton control styles.....	369
RadioButton control functions and statements.....	370
Notification messages.....	371
Rebar Controls - WIP	372
About Rebar.....	372
Creating the control.....	372
Rebar control styles.....	372
Rebar control functions and statements.....	373
Notification messages.....	374
RichEdit Controls - WIP	374
About rich edit.....	374
Creating the control.....	375
Rich edit control styles.....	375
Controlling the rich edit control.....	377
Clipboard operations.....	377
Line operations.....	377
Selection operations.....	378
Formatting operations.....	378
Editing operations.....	379
General operations.....	380
Notification event control.....	382
Notification messages.....	382
Miscellaneous.....	383
Scrollbar Controls - WIP	384
About scroll bar.....	384
Creating the control.....	385
Scroll bar control styles.....	385
Scroll bar control functions and statements.....	385
Messages.....	387
Notification messages.....	388
Spinner Controls - WIP	389
About Spinner.....	389
Creating the control.....	389
Spinner control styles.....	389
Spinner control functions and statements.....	391
Notification messages.....	392

Static Controls - WIP	393
About Static.....	393
Creating the control.....	393
Static control styles.....	393
SysButton control functions and statements	394
Notification messages.....	395
Status Controls - WIP	396
About status windows.....	396
Creating the control.....	396
Controlling the status window control.....	396
Statements and Functions.....	396
Tab Controls - WIP	398
About Tab.....	398
Creating the control.....	398
Tab control styles.....	398
Tab control functions and statements.....	400
Notification messages.....	402
ToolBar Controls - WIP	403
About Toolbar controls.....	403
Creating the control.....	403
ToolBar control styles.....	403
Controlling the toolbar control.....	405
Statements and Functions.....	405
Design Considerations.....	407
ToolTip Controls - WIP	407
About ToolTip.....	407
Creating the control.....	407
ToolTip control styles.....	407
ToolTip control functions and statements.....	408
Notification messages.....	409
TrackBar Controls - WIP	410
About Trackbar.....	410
Creating the control.....	410
Trackbar control styles.....	410
Trackbar control functions and statements.....	411
Notification messages.....	413
TreeView Controls - WIP	413
About tree view	413
Creating the control.....	414
Tree view control styles.....	414
Statements and functions for controlling the tree view	415
Notification messages.....	417
UDT's used with the tree view control.....	419

Part XI How-To

422

1 Set Startup Preferences	422
On Startup	423
On Project Open	423
On File Open	423
Recent Files/Projects	423
2 Set Editor Preferences	424
3 Set Compiler Preferences	430
Flags	431

Defines	432
Include Dir	434
Additional Lib Dir	435
Command Paks	437
4 Files.....	438
Create a File	438
Open a File	438
Save a File	442
Close a File	442
Find Text in a File	444
Replace Text in a File	445
5 Single File Application.....	445
Set Application Options	446
Compile an Application	449
Run an Application	450
6 Projects.....	451
Create a New Project	451
Set Project Options	454
Open a Project	457
Close a Project	462
Save a Project	462
Add Files	463
*.iw b files.....	463
*.obj files.....	464
*.asm files.....	464
*.lib files	465
Remove Files	465
Compile a Project	466
Run a Project	468
Resources	469
Compile a Resource	475
7 Convert from CBasic / IBasic.....	476
Why convert to IWBASIC?	476
Getting Started	476
Subroutines - Local	476
Subroutines - Components	479
Subroutines - DLL Calls	481
Windows / Dialogs	482
Controls	483
Menus	484
Menus - Context	485
INSTR	486
ListView	476
2D/3D Graphics	486
Conclusion	487
8 Convert from EBasic / IBPro.....	487

Part XII Alphabetical Command Reference 490

1 ABS.....	490
2 ADDACCELERATOR.....	490
3 ACOS.....	491

4	ACOSD.....	491
5	ADDMENUITEM.....	492
6	ADDSTRING.....	492
7	ALIAS.....	493
8	AllocHeap.....	493
9	ALLOCMEM.....	494
10	APPEND\$.....	495
11	APPENDMENU.....	495
12	ASC.....	496
13	ASIN.....	496
14	ASIND.....	497
15	ATAN.....	497
16	ATAND.....	497
17	ATTACHBROWSER.....	498
18	AUTODEFINE.....	499
19	BACKPEN.....	499
20	BASELEN.....	500
21	BEGININSERTMENU.....	500
22	BEGINMENU.....	501
23	BEGINPOPUP.....	502
24	BFILE.....	502
25	BREAK.....	503
26	BREAKFOR.....	503
27	BROWSECMD.....	504
28	BYTE.....	505
29	CalendarControl.....	505
30	CALLOBJECTMETHOD.....	506
31	CASE&.....	507
32	CASE.....	508
33	CATCH.....	508
34	cbeAddString.....	509
35	cbeDeleteString.....	509
36	cbeGetSelected.....	510
37	cbeGetString.....	510
38	cbeGetStringCount.....	511
39	cbeInsertString.....	511
40	cbeSetImageList.....	512
41	cbeSetIndent.....	513
42	cbeSetSelected.....	513

43	ccGetColor.....	514
44	ccGetCurSel.....	514
45	ccGetFirstDayOfWeek.....	515
46	ccGetMinimumRect.....	515
47	ccGetScrollDelta.....	516
48	ccGetToday.....	516
49	ccSetColor.....	517
50	ccSetCurSel.....	518
51	ccSetFirstDayOfWeek.....	518
52	ccSetScrollDelta.....	519
53	ccSetToday.....	519
54	CEIL.....	520
55	CENTERWINDOW.....	520
56	CHAR.....	520
57	CHECKMENUITEM.....	521
58	CHR\$ / WCHR\$.....	521
59	CIRCLE.....	522
60	CLOSECONSOLE.....	523
61	CLOSEDIALOG.....	523
62	CLOSEFILE.....	524
63	CLOSEPRINTER.....	524
64	CLOSEWINDOW.....	525
65	CLS.....	525
66	COLOR.....	525
67	COLORREQUEST.....	526
68	ComboBoxEx.....	527
69	COMENUMBEGIN.....	527
70	COMENUMNEXT.....	528
71	COMREF.....	529
72	CONST.....	529
73	CONTEXTMENU.....	530
74	CONTROL.....	531
75	CONTROLCMD.....	532
76	CONTROLEX.....	532
77	CONTROLEXISTS.....	533
78	COPYFILE.....	533
79	COPYRGN.....	534
80	COS.....	534
81	COSD.....	535

82	COSH.....	535
83	COSHD.....	536
84	CREATECOMOBJECT.....	536
85	CREATEDIALOG.....	537
86	CREATEDIR.....	537
87	CREATEMENU.....	538
88	CREATEREGKEY.....	538
89	DATA.....	539
90	DATABEGIN.....	540
91	DATAEND.....	540
92	DATE\$.....	541
93	DateTimePicker.....	542
94	DEBUGPRINT.....	543
95	DECLARE.....	543
96	DEF / DIM.....	544
97	DEFAULT.....	545
98	DEFINE_GUID.....	546
99	DELETE.....	546
100	DELETEFILE.....	547
101	DELETEIMAGE.....	547
102	DELETEREGKEY.....	548
103	DELETERGN.....	548
104	DELETESTRING.....	549
105	DICTADD.....	549
106	DICTCREATE.....	550
107	DICTFREE.....	551
108	DICTGETKEY.....	551
109	DICTGETNEXTASSOC.....	552
110	DICTGETSTARTASSOC.....	552
111	DICTGETVALUE.....	553
112	DICTLOOKUP.....	553
113	DICTREMOVE.....	554
114	DICTREMOVEALL.....	554
115	DO.....	555
116	DOMODAL.....	555
117	DOUBLE.....	556
118	DRAWMODE.....	556
119	dtpGetMCColor.....	557
120	dtpGetSystemTime.....	557

121	dtpSetFormat.....	558
122	dtpSetMCColor.....	559
123	dtpSetSystemTime.....	560
124	EACH.....	561
125	ELLIPSE.....	561
126	ELSE.....	562
127	ELSEIF.....	562
128	ENABLECONTROL.....	563
129	ENABLEMENU.....	563
130	ENABLEMENUITEM.....	564
131	ENABLETABS.....	564
132	END.....	565
133	ENDCATCH.....	565
134	ENDENUM.....	566
135	ENDIF.....	567
136	ENDINTERFACE.....	567
137	ENDMENU.....	568
138	ENDPAGE.....	568
139	ENDPOPOP.....	569
140	ENDSELECT.....	569
141	ENDSUB.....	570
142	ENDTRY.....	571
143	ENDTYPE.....	571
144	ENDUNION.....	572
145	ENDWHILE.....	573
146	ENUM.....	573
147	EOF.....	574
148	EXP.....	575
149	EXPORT.....	575
150	EXTERN.....	576
151	FACOS.....	576
152	FACOSD.....	577
153	FASIN.....	577
154	FASIND.....	578
155	FATAN.....	578
156	FATAND.....	579
157	FCOS.....	579
158	FCOSD.....	580
159	FCOSH.....	580

160	FCOSHD.....	581
161	FILE.....	581
162	FILEREQUEST.....	581
163	FINDCLOSE.....	583
164	FINDNEXT.....	583
165	FINDOPEN.....	585
166	FLOODFILL.....	586
167	FLOOR.....	586
168	FLT / FLOAT.....	587
169	FONTREQUEST.....	587
170	FOR.....	588
171	FreeHeap.....	589
172	FREELIB.....	589
173	FREEMEM.....	590
174	FRONTPEN.....	590
175	FSIN.....	591
176	FSIND.....	591
177	FSINH.....	592
178	FSINHD.....	592
179	FTAN.....	593
180	FTAND.....	593
181	FTANH.....	594
182	FTANHD.....	594
183	GET.....	595
184	GETBITMAPSIZE.....	595
185	GETCAPTION.....	596
186	GETCARETPOSITION.....	596
187	GETCLIENTSIZE.....	597
188	GETCOMPROPERTY.....	597
189	GETCONTROLHANDLE.....	599
190	GETCONTROLTEXT.....	599
191	GETDATA.....	600
192	GETDEFAULTPRINTER.....	600
193	GETEXCEPTIONCODE.....	601
194	GETEXCEPTIONINFORMATION.....	601
195	GETFOLDERPATH.....	602
196	GETHDC.....	604
197	GETKEYSTATE.....	604
198	GETPIXEL.....	605

199	GETPOSITION.....	605
200	GetProgressPosition.....	606
201	GETRESOURCELENGTH.....	606
202	GETSCREENSIZE.....	607
203	GETSCROLLPOS.....	608
204	GETSCROLLRANGE.....	608
205	GETSELECTED.....	609
206	GETSIZE.....	609
207	GetSpinnerBase.....	610
208	GetSpinnerBuddy.....	610
209	GetSpinnerPosition.....	611
210	GetSpinnerRangeMax.....	611
211	GetSpinnerRangeMin.....	612
212	GETSTARTPATH.....	612
213	GETSTARTPATHW.....	613
214	GETSTATE.....	613
215	GETSTRING.....	614
216	GETSTRINGCOUNT.....	614
217	GETTEXTSIZE.....	614
218	GETTHUMBPOS.....	615
219	GetTrackBarLineSize.....	616
220	GetTrackBarPageSize.....	616
221	GetTrackBarPosition.....	617
222	GetTrackBarRangeMax.....	617
223	GetTrackBarRangeMin.....	618
224	GLOBAL.....	618
225	GOSUB.....	619
226	GOTO.....	619
227	HeaderControl.....	620
228	HeapClear.....	621
229	HEX\$ / WHEX\$.....	621
230	hcDeleteItem.....	622
231	hcGetItemCount.....	622
232	hcGetItemData.....	623
233	hcGetItemRect.....	623
234	hcGetItemText.....	624
235	hcGetItemWidth.....	624
236	hcInsertItem.....	625
237	hcSetImageList.....	625

238	hcSetItemData.....	626
239	hcSetItemJustify.....	626
240	hcSetItemText.....	627
241	hcSetItemWidth.....	628
242	IF	628
243	IMPORT.....	629
244	INKEY\$.....	629
245	INPUT.....	630
246	INSERTMENU.....	631
247	INSERTSTRING.....	631
248	INSTR.....	632
249	INT	632
250	INT64.....	633
251	INTERFACE.....	633
252	IPClearAddress.....	634
253	IPControl.....	634
254	IPGetAddress.....	635
255	IPGetAddressDword.....	635
256	IPIsBlank.....	636
257	IPSetAddress.....	636
258	IPSetAddressDword.....	637
259	IPSetRange.....	637
260	ISREF.....	638
261	ISSELECTED.....	638
262	ISWINDOWCLOSED.....	639
263	LABEL.....	639
264	LCASE\$ / WLCASE\$.....	640
265	LEAVE.....	640
266	LEFT\$ / WLEFT\$.....	641
267	LEN.....	641
268	LINE.....	642
269	LINETO.....	643
270	ListAdd.....	643
271	ListAddHead.....	644
272	ListCreate.....	644
273	ListGetData	645
274	ListGetFirst.....	646
275	ListGetNext.....	646
276	ListRemove.....	647

277	ListRemoveAll.....	647
278	LOADIMAGE.....	648
279	LOADMENU.....	649
280	LOADRESOURCE.....	650
281	LOADTOOLBAR.....	651
282	LOCATE.....	651
283	LOG10.....	652
284	LOG.....	653
285	LTRIM\$ / WLTRIM\$.....	653
286	MEMORY.....	654
287	MENUITEM.....	654
288	MENUTITLE.....	655
289	MESSAGEBOX.....	655
290	MID\$ / WMID\$.....	657
291	MILLISECS.....	657
292	MODIFYEXSTYLE.....	658
293	MODIFYSTYLE.....	658
294	MOVE.....	659
295	NEW.....	659
296	NEXT.....	660
297	NOT.....	661
298	ONCONTROL.....	661
299	ONMENUPICK.....	663
300	ONMESSAGE.....	664
301	ONEXIT.....	666
302	OPENCONSOLE.....	667
303	OPENFILE.....	667
304	OPENPRINTER.....	668
305	OPENWINDOW.....	669
306	PagerControl.....	669
307	pcSetPos.....	670
308	pcSetChildHwnd.....	671
309	pcSetChild.....	671
310	pcSetButtonSize.....	672
311	pcSetBorderSize.....	672
312	pcSetBackColor.....	673
313	pcRecalcSize.....	673
314	pcGetPos.....	674
315	pcGetButtonState.....	675

316	pcForwardMouse.....	675
317	PLAYMIDI\$.....	676
318	PLAYWAVE.....	676
319	POWER.....	677
320	PRINT.....	677
321	PRINTWINDOW.....	678
322	ProgressControl.....	679
323	ProgressStepIt.....	680
324	PROJECTGLOBAL.....	680
325	PRTDIALOG.....	681
326	PSET.....	682
327	PushHeap.....	682
328	PUT.....	683
329	RAND.....	683
330	RASTERMODE.....	684
331	rbAddBand.....	684
332	rbSetBandBitmap.....	685
333	rbSetBandChild.....	686
334	rbSetBandChildHandle.....	686
335	rbSetBandColors.....	687
336	rbSetBandText.....	688
337	rbShowBand.....	688
338	READ.....	689
339	READMEM.....	689
340	RebarControl.....	690
341	RECT.....	691
342	REDRAWFRAME.....	692
343	REGGETDWORD.....	692
344	REGGETSTRING.....	693
345	REGSETDWORD.....	693
346	REGSETSTRING.....	694
347	RELEASEHDC.....	694
348	REMOVEDIR.....	695
349	REMOVEMENUITEM.....	695
350	REPLACE\$.....	696
351	RESTORE.....	696
352	RETURN.....	697
353	RGB.....	698
354	RGNFROMBITMAP.....	698

355	RIGHT\$ / WRIGHT\$.....	699
356	RND.....	700
357	RTRIM\$ / WRTRIM\$.....	700
358	S2W.....	701
359	SCHAR.....	701
360	SEEDRND.....	702
361	SEEK.....	702
362	SELECT.....	703
363	SENDMESSAGE.....	703
364	SEPARATOR.....	704
365	SET_INTERFACE.....	704
366	SETBUTTONBITMAPS.....	705
367	SETBUTTONBORDER.....	706
368	SETBUTTONRGN.....	707
369	SETCAPTION.....	707
370	SETCOMPPROPERTY.....	708
371	SETCONTROLCOLOR.....	709
372	SETCONTROLNOTIFY.....	709
373	SETCONTROLTEXT.....	710
374	SETCURSOR.....	711
375	SETEXITCODE.....	711
376	SETFOCUS.....	712
377	SETFONT.....	712
378	SETHORIZEXTENT.....	714
379	SETHTCOLOR.....	714
380	SETICON.....	715
381	SETID.....	715
382	SETLBCOLWIDTH.....	716
383	SETLINESTYLE.....	716
384	SETMENU.....	717
385	SETPRECISION.....	717
386	SetProgressBarColor.....	718
387	SetProgressDelta.....	718
388	SetProgressMarquee.....	719
389	SetProgressPosition.....	720
390	SetProgressRange.....	720
391	SetProgressStep.....	721
392	SETSCROLLPOS.....	721
393	SETSCROLLRANGE.....	722

394	SETSELECTED.....	722
395	SETSIZE.....	723
396	SetSpinnerBase.....	723
397	SetSpinnerBuddy.....	724
398	SetSpinnerPosition.....	724
399	SetSpinnerRange.....	725
400	SETSTATE.....	725
401	SetTrackBarLineSize.....	726
402	SetTrackBarPageSize.....	727
403	SetTrackBarPosition.....	727
404	SetTrackBarRange.....	728
405	SetTrackBarThumbLength.....	728
406	SetTrackBarTickFreq.....	729
407	SETTYPE.....	729
408	SETWINDOWCOLOR.....	730
409	SGN.....	731
410	SHOWCONTEXTMENU.....	731
411	SHOWDIALOG.....	732
412	SHOWIMAGE.....	732
413	SHOWWINDOW.....	733
414	SIN.....	734
415	SIND.....	734
416	SINH.....	734
417	SINH.....	735
418	SIZEOF.....	735
419	SPACE\$ / WSPACE\$.....	736
420	SpinnerControl.....	736
421	SQRT.....	737
422	STARTTIMER.....	738
423	STATIC.....	739
424	STDMETHOD.....	739
425	STEP.....	740
426	STOP.....	740
427	STOPMIDI\$.....	741
428	STOPTIMER.....	741
429	STR\$ /WSTR\$.....	742
430	STRING\$ /WSTRINGS\$.....	742
431	SUB.....	743
432	SWORD.....	744

433	SYSTEM.....	744
434	TabControl.....	745
435	TAN.....	746
436	TAND.....	747
437	TANH.....	747
438	TANHD.....	747
439	tcDeleteAllTabs.....	748
440	tcDeleteTab.....	748
441	tcGetFocusTab.....	749
442	tcGetItemData.....	749
443	tcGetRowCount.....	750
444	tcGetSelectedTab.....	750
445	tcGetTabCount.....	751
446	tcGetTabText.....	751
447	tcHighlightTab.....	752
448	tcHitTest.....	752
449	tcInsertTab.....	753
450	tcSetFocusTab.....	753
451	tcSetImage.....	754
452	tcSetImageList.....	754
453	tcSetItemData.....	755
454	tcSetMinTabSize.....	755
455	tcSetSelectedTab.....	756
456	tcSetTabText.....	756
457	tcSetTip.....	757
458	THEN.....	757
459	THREAD.....	758
460	THROW.....	758
461	TIME\$.....	759
462	TIMER.....	759
463	TO.....	760
464	ToolTipControl.....	760
465	TrackBarControl.....	761
466	TRY.....	762
467	ttSetToolRect.....	763
468	ttRelayMessage.....	763
469	ttDeleteTool.....	764
470	ttAddTool.....	765
471	tvDeleteAllItems.....	766

472	tvDeleteItem.....	766
473	tvGetItemData.....	766
474	tvGetItemText.....	767
475	tvGetSelectedItem.....	768
476	tvInsertItem.....	768
477	tvSelectItem.....	769
478	tvSetItemData.....	769
479	tvSetItemText.....	770
480	TYPE.....	770
481	TYPEOF.....	771
482	UCASE\$ / WUCASE\$.....	772
483	UINT.....	772
484	UINT64.....	773
485	UNION.....	773
486	UNTIL.....	774
487	USING / WUSING.....	774
488	VAL.....	775
489	W2S.....	775
490	WAIT.....	776
491	WAITCON.....	776
492	WAITUNTIL.....	777
493	WEND.....	778
494	WHILE.....	778
495	WORD.....	779
496	WRITE.....	779
497	WRITEMEM.....	780
498	WRITEPRINTER.....	780

Part XIII Appendix - User's Guide

784

1	A. Compiler preprocessor reference.....	784
	_asm.....	784
	_endasm.....	784
	#DEFINE.....	784
	\$COMMAND.....	784
	\$ASM.....	784
	\$DEFINE.....	784
	\$ELSE.....	784
	\$ELIF.....	784
	\$ELIFDEF.....	784
	\$ELIFNDEF.....	785
	\$EMIT.....	785
	\$END.....	785
	\$ENDASM.....	785

\$ENDIF	785
\$ENDREGION	785
\$ERROR	785
\$IF	785
\$IFDEF	785
\$IFNDEF	785
\$INCLUDE	785
\$MACRO	785
\$MAIN	785
\$OPTION	785
\$REGION	785
\$THREAD	786
\$TYPEDEF	786
\$UNDECLARE	786
\$UNDEF	786
\$USE	786
\$WARNING	786
'ENDREGION	786
'REGION	786
AUTODEFINE	784
CONST	786
EXPORT	786
EXTERN	786
GLOBAL	786
PROJECTGLOBAL	786
SETID	787
SET_INTERFACE	787
SETTYPE	787
2 B. Message variables, ID's and constants	787
3 C. Virtual key codes	802
4 D. ASCII table	805
5 E. Special Constants	807
Part XIV 2D Programming Guide	810
1 Introduction	810
2 Creating and Using Screens	810
3 Graphic Commands	812
4 Using Sprites	814
5 Collision Detecting	819
6 Scrolling Tile Maps	820
7 Mouse and Keyboard Input	823
8 Joysticks and Gamepads	825
9 Using 8 bpp Screens	827
10 Windowed mode	829
11 Direct Buffer/Sprite Writing	830
12 Alphabetical Command Reference	832
ATTACHSCREEN	835
CLOSESCREEN	835

CREATEMAPDATA	836
CREATESCREEN	837
CreateSprite	837
Draw AALine	838
Draw AlphaLine	839
Draw FilledRect	839
Draw Line	840
DRAWMAP	840
Draw Rect	841
Draw Sprite	841
Draw SpriteXY	842
FADEPALETTE	842
FILLPALETTE	843
FILLSCREEN	844
FLIP	844
FLUSHKEYS	845
FREEMAP	845
FreeSprite	846
GetBufferHeight	846
GetBufferPitch	847
GetBufferPointer	848
GetBufferWidth	849
GETJOYSTICKAXISCOUNT	849
GETJOYSTICKBUTTONCOUNT	849
GETJOYSTICKCOUNT	850
GETJOYSTICKNAME	850
GETJOYSTICKTYPE	851
GETKEY	851
GETMAPCOUNT	852
GETMAPDATA	852
GETMAPHEIGHT	853
GETMAPPIXELHEIGHT	853
GETMAPPIXELWIDTH	854
GETMAPWIDTH	854
GETPALETTECOLOR	855
GetSpriteDelay	855
GetSpriteFrames	856
GetSpriteHeight	856
GetSpritePitch	857
GetSpritePointer	857
GetSpriteState	858
GetSpriteType	859
GetSpriteVelX	859
GetSpriteVelY	860
GetSpriteWidth	860
JOYDOWN	861
JOYX	861
JOYY	862
JOYZ	862
KEYDOWN	863
LOADMAPDATA	863
LOADPALETTE	864
LoadSprite	864
LOCKBUFFER	865

LOCKSPRITE	866
MAPDRAWMODE	867
MAPMASKCOLOR	868
MOUSEDOWN	868
MOUSEX	869
MOUSEY	869
MOVEMAP	870
MoveSprite	870
NEWMAP	871
PALETTEINDEX	871
ReadPixel	872
RGBToScreen	873
SAVEMAPDATA	873
SCROLLMAP	874
SETJOYSTICKDEADZONE	875
SETJOYSTICKRANGE	875
SETMAPDATA	876
SETMAPVIEWPORT	877
SETPALETTECOLOR	877
SetSpriteDelay	878
SetSpriteState	878
SetSpriteType	879
SetSpriteVelX	879
SetSpriteVelY	880
SpriteAlpha	880
SpriteAngle	881
SpriteCollided	881
SpriteCollidedEx	882
SpriteDraw Mode	883
SpriteFrame	884
SpriteMaskColor	884
SpriteScaleFactor	885
SpriteShadow Offset	885
SpriteToBuffer	886
UNLOCKBUFFER	887
UNLOCKSPRITE	887
WAITKEY	888
WriteAlphaPixel	888
WritePixel	889
WritePixelFast	889
WriteText	890
13 Appendix	891
DirectInput Keyboard Codes	891

Part XV 3D Programming Guide 898

1 Introduction.....	898
2 Classes.....	898
C3DCamera	898
_C3DCamera.....	899
C3DCamera.....	899
Create	900
EnableFog.....	900
Free	901

GetDirection.....	901
GetLookAt.....	902
GetPosition.....	902
GetUpVector.....	902
LockYAxis.....	903
LookAt	903
Move	904
ObjectInView	904
Orient	905
Position	905
Project	906
Rotate	906
SetAspectRatio.....	907
SetBackPlane.....	907
SetFogColor.....	908
SetFogRange.....	908
SetFOV	909
SetFrontPlane.....	909
SetMode	910
SetY	910
Unproject.....	911
C3DLandscape	911
Load	912
C3DLight	913
Create	914
Disable	915
Enable	916
SetAmbient.....	916
SetAttenuation.....	917
SetDiffuse.....	918
SetDirection.....	919
SetFalloff.....	920
SetPhi	920
SetRange.....	921
SetSpecular.....	922
SetTheta.....	922
C3DMesh	923
BeginRenderCubeTexture.....	925
BuildOctree.....	926
CreateBox.....	927
CreateCubeTexture.....	927
CreateCylinder.....	929
CreateMesh.....	930
CreateMeshEx.....	931
CreateRectangle.....	932
CreateSphere.....	932
EnableAlpha.....	933
EnableLighting.....	934
EnableSphereMapping.....	934
EndRenderCubeTexture.....	935
GetID	936
GetIndexCount.....	936
GetVertexCount.....	937
GetVertexFormat.....	937

GetVertexSize.....	938
Load3DS.....	939
LoadMD2.....	939
LoadSkinnedX.....	940
LoadTexture.....	941
LoadX.....	942
LockIndexBuffer.....	942
LockVertexBuffer.....	943
ReallocateMesh.....	944
RecalcBoundingBox.....	945
SetAlphaArg1.....	945
SetAlphaArg2.....	946
SetAlphaDest.....	947
SetAlphaOp.....	948
SetAlphaOperation.....	949
SetAlphaSource.....	949
SetAnimation.....	950
SetAnimationMode.....	951
SetColorArg1.....	952
SetColorArg2.....	953
SetColorOperation.....	954
SetCulling.....	954
SetFill.....	955
SetID.....	956
SetMaterial.....	956
SetNamedAnimation.....	957
SetShading.....	958
SetVertexFormat.....	959
SetVertexSize.....	960
SetVisible.....	960
UnlockIndexBuffer.....	961
UnlockVertexBuffer.....	962
UpdateAllAnimations.....	962
UpdateAnimation.....	963
UseVertexColor.....	964
C3DObject.....	964
_C3DObject.....	965
AddChild.....	966
C3DObject.....	966
CreateScene.....	967
CreateTransform.....	967
Draw.....	968
Free.....	969
GetCollisionPoint.....	969
GetDirection.....	970
GetMatrix.....	970
GetPosition.....	971
InitCollision.....	971
LookAt.....	972
ObjectCollided.....	973
Orient.....	974
Position.....	974
RayCollided.....	975
Rotate.....	976

Scale	976
SetMatrix.....	977
SphereCollided.....	977
C3DScreen	978
_C3DScreen.....	979
Begin2D	979
BeginScene.....	980
C3DScreen.....	981
Clear	981
CloseScreen.....	982
CreateFullScreen.....	982
CreateWindow ed.....	983
End2D	984
MouseX	985
MouseY	986
RenderScene.....	986
RenderText.....	987
Reset	988
SetFont	988
SetRestoreCallback.....	989
C3DSprite	989
_C3DSprite.....	990
C3DSprite.....	991
Draw	991
Free	992
GetAngle.....	992
GetFrame.....	993
GetModulateColor.....	993
GetPosition.....	994
GetRotationCenter.....	994
GetScaleFactor.....	995
Load	995
SetAngle.....	996
SetFrame.....	997
SetModulateColor.....	997
SetPosition.....	998
SetRotationCenter.....	998
SetScaleFactor.....	999
3 Structures.....	999
D3DCOLORVALUE	999
D3DMATERIAL	1000
MATRIX4	1000
VECTOR2	1000
VECTOR3	1000
VECTOR4	1000
VERTEX0TEXTURE	1001
VERTEX1TEXTURE	1001
VERTEX2TEXTURE	1001
VERTEX3TEXTURE	1001
VERTEX4TEXTURE	1002
VERTEX5TEXTURE	1002
VERTEX6TEXTURE	1002
VERTEX7TEXTURE	1003
VERTEX8TEXTURE	1003

4	Functions.....	1003
	RGBA	1003
	Vec3Add	1004
	Vec3Cross	1004
	Vec3Dot	1005
	Vec3Length	1005
	Vec3Lerp	1005
	Vec3Normalize	1006
	Vec3Sub	1007
	Vec4Add	1007
	Vec4Cross	1007
	Vec4Dot	1008
	Vec4Length	1008
	Vec4Lerp	1009
	Vec4Normalize	1009
	Vec4Sub	1010
	MatrixIdentity	1010
	MatrixTranslation	1011
	MatrixRotation	1011
	MatrixMultiply	1011
5	Global_Constants.....	1012
	Alpha Blending Constants	1012
	Alpha Operator Constants	1013
	Animation Modes	1013
	Culling Flags	1013
	Flexible Vertex Format Constants	1013
	Light Types	1014
	Mesh Fill Styles	1014
	Mesh Shading Styles	1014
	Primitive Types	1015
	Texture Blending Constants	1015

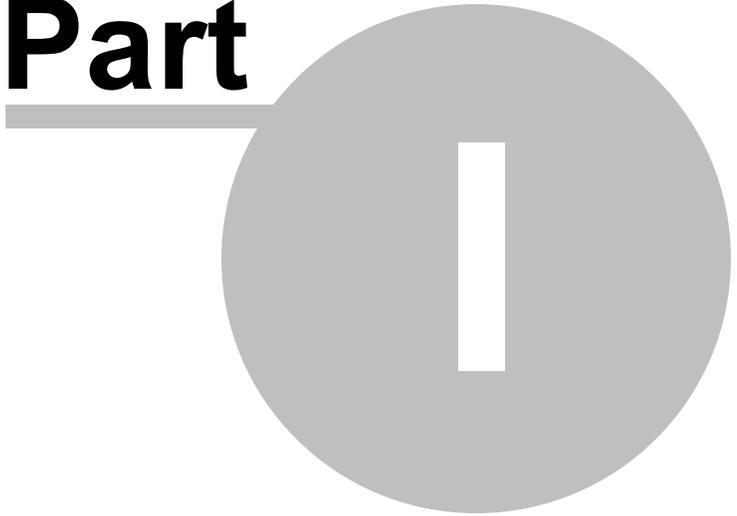
Part XVI Database Programming Guide 1020

1	Introduction.....	1020
2	Connecting to the database.....	1020
3	Catalog Functions.....	1022
4	SQL.....	1023
5	Retrieving Results.....	1027
6	Updates and Insertions.....	1031
7	Accessing the ODBC API.....	1033
8	Alphabetical Command Reference.....	1034
	dbBindDate	1034
	dbBindDateParam	1035
	dbBindParameter	1036
	dbBindTime	1037
	dbBindTimeParam	1038
	dbBindTimeStamp	1039
	dbBindTimeStampParam	1040
	dbBindVariable	1040
	dbCardinality	1042

dbConnect	1042
dbConnectDSN	1043
dbCreateMDB	1043
dbDisconnect	1044
dbEnum Drivers	1044
dbExecSQL	1045
dbExecute	1046
dbFreeSQL	1047
dbGet	1047
dbGetData	1048
dbGetDate	1049
dbGetErrorCode	1050
dbGetErrorText	1051
dbGetFirst	1051
dbGetLast	1052
dbGetNext	1054
dbGetNum Cols	1055
dbGetPrev	1056
dbGetTime	1057
dbGetTimeStamp	1058
dbIs Null	1059
dbListColumns	1059
dbListTables	1060
dbPrepareSQL	1061
9 Appendix.....	1062
Minimum SQL Grammar	1062
Part XVII Disclaimer / License / Copyright	1066
Part XVIII History	1070

Introduction

Part



1 Introduction

IW BASIC is a full featured 32 bit compiler for the Windows™ operating system. Capable of producing small, fast executables and DLL's. IW BASIC is the perfect language for any programming task.

Main Features:

- Fast 32 bit assembler, linker, and compiler.
- Uses standard COFF and LIB format files.
- Easily upgradeable and expandable.
- Integrated Scintilla based editor and debugger.
- Multi module programming through projects.
- Supports Windows 98,ME,NT,2000,XP,Vista, and Win7.
- Executables created are royalty free.
- Able to create and use static libraries.
- Generates native machine code, no runtimes!

Language Features:

- BASIC like syntax.
- Rich command set, over 800 built in commands and functions.
- Extensive mathematic operators and functions.
- Built in linked-list handling.
- Easy window, dialog and control creation.
- Advanced inline assembler for optimizing code.
- Use IWBasic variables directly in inline assembly code.
- Interface IWBasic variables with assembly raw data.
- Text console support.
- Advanced and easy pointer operators.
- 'C' style pointer operations.
- Built in midi music and sound commands.
- Case insensitive keywords.
- Optional casesensitive variable names.
- Graphics primitive operations for quick and simple drawing operations.
- Text and graphics printer output support.
- Web enabled application development with the integrated HTML browser control.
- Simple common control commands and functions.
- Quickly interfaces with the Windows API, C runtime libraries, and static code libraries.
- Automatically creates missing import libraries, when possible.
- Optional, default, and variable number of parameters for functions.
- Static variables in subroutines/functions.
- Supports indirect function calls.
- ANSI compliant UDT and UNION types (structure).
- Nested structure definitions.
- STDCALL and CDECL function calling conventions supported.
- Import variables and methods from DLL files.

- OOP fully supported.
- Structured exceptions handling.
- Built in ODBC database support.
- Built in 2D gaming commands.
- DirectX 9.0 3D gaming engine included.
- Built in Unicode support with ability to code both ANSI/Unicode under flag control.
- Built in COM support.
- Built in support for resources.
- Supports macro creation with multiple expressions.
- Supports thread-private variables.
- Built in Drag and Drop commands.
- IBasic™ Professional Compatible.
- Emergence Basic™ Compatible
- Help file in chm, pdf, and eBook formats.

Minimum Requirements for IDE:

- Pentium 200 or better processor.
- 64MB of ram.
- 30MB Free hard drive space.
- Windows 98 or greater -or- Windows 2000 or greater.

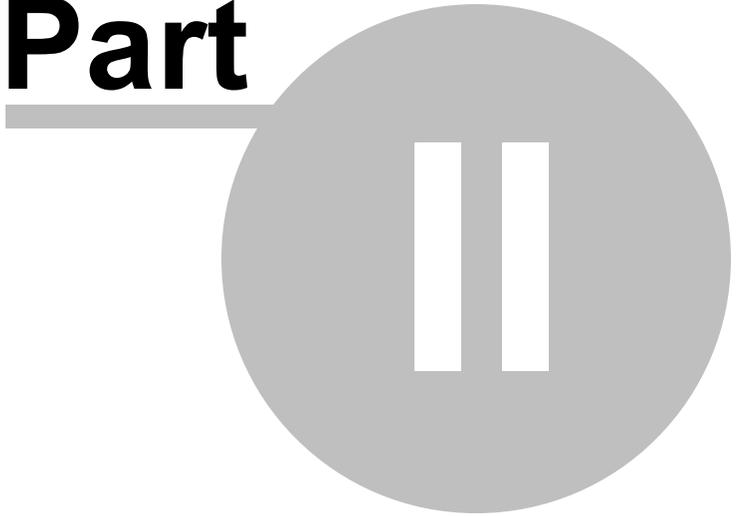
Redistributable software notice

The assembler used by the IW BASIC development environment is a derivation of the Netwide Assembler (NASM) which is distributed under the GNU LGPL (Lesser General Public License). The full source code archive to the derived assembler can be found in the *redist* directory under the main installation directory. Please refer to the "copying" file in the archive for information on use of the source code for the assembler.

The LGPL license only applies to the assembler and it's source code, not any programs you make with the IW BASIC development environment, nor does it apply to any other libraries or executables included with the IW BASIC development environment which are Copyright © 2011 Ionic Wind Software. Refer to subsection 5 of the LGPL for more information.

Getting Started: Beginners

Part



2 Getting Started: Beginners

If you are new to programming you should read this section before proceeding.

Generally speaking there are a standard series of steps a programmer goes through in creating an application, regardless of the programming language being used. These three simple steps are:

1. Write the program, using the selected language syntax, in one or more text files.
2. Invoke a language parser and assembler to convert the text files into machine code files.
3. Invoke a linker and pass all the machine code files and required library files to the linker.

The most basic level of accomplishing the three steps would be as follows:

1. Open NotePad (or some other text editor of choice)
2. Write your code and save the text file.
3. Repeat as many times as needed for multiple source files.
4. At the dot prompt enter the name of the parser plus any required command line options plus the name of the text file. Hit enter.
5. Repeat for each of the source files.
6. If there is a failure then go back to step #1, fix the problem in your code, and try again.
7. For multiple source files the process can be sped up by creating a batch file with multiple #4 lines in it.
8. Create a "make" file for the linker that contains all your machine code files created by the parser/assembler plus all the library files that the language uses internally plus any that you are using.
9. At the dot prompt enter the name of the linker plus any required command line options plus the name of the "make" file. Hit enter.
10. Run the resulting executable if you have one. If the link failed then fix the problem, either by going back to step #1 or #8, and try again.

Some programs use resource files. If yours does then you will need to perform steps #1 - #6 with the resource file and the resource compiler. The resulting machine code file will have to be added to the "make" file in step #8.

All the above assumes you are only using static libraries. There are times when some third party has created a Dynamic Link Library that we just have to use. In that case you will need to invoke yet another program to read the DLL and create a linking library whose name you will add to the "make" file in step #8.

To be honest, I don't think there are many, if any, people who program that way anymore.

So, the obvious question is how do we get around all those steps above?

We use an Integrated Development Environment (IDE). What is an IDE?

A basic IDE is a program that contains a text editor and a toolbar/menu that has functions that ultimately perform all the steps outlined above. Once a user has finished writing all the code for a program the simple clicking of a button will cause an executable file to be created. Furthermore, the IDE contains other useful utility functions designed to aid the programmer in a wide range of ways.

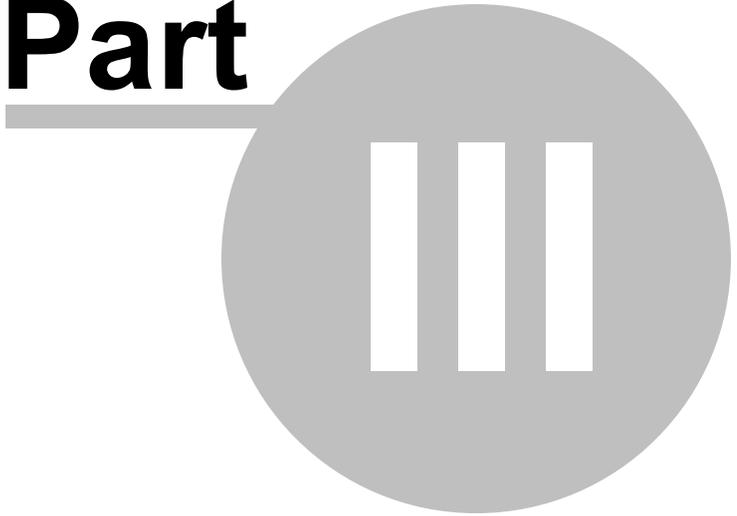
We suggest that you now take the time to at least scan the [IDE Interface](#) section of this help file before attempting to create a program.

Enjoy,

The IonicWind Software Team

Getting Started: Upgrading

Part



3 Getting Started: Upgrading

This section is intended for new Users of IWBasic who are upgrading from IBasic, IBasic Professional (IBPro), Emergence Basic (EBasic), and Creative Basic (CBasic).

History

IWBasic's roots can be traced back twenty years to the IBasic language. IBasic was an interpreter. The language had a strong following and there were a lot of programs written for it. The author of IBasic then created a second language called IBasic Professional. This language was a compiled language and, although similar to the original IBasic, it was not backward compatible. This resulted in a split in the User base but both languages continued to thrive.

The author then decided to sell his rights to the two languages. The new owner almost immediately abandoned the two languages and shut down the support forum. There was such a clamor from the User base that the original author decided to create two new languages with the intent that they be, as completely as possible, backward compatible with the two original languages.

Creative Basic (CBasic), the interpreter, was written to support the IBasic Users and Emergence Basic (EBasic), the compiler, was written to support the IBPro Users. This effort was spread over a period of time and programs continued to be written for both languages. After a while the future of the two new languages became uncertain and the languages were sold. After some additional sales the current owner obtained both languages, at different times.

EBasic was extensively reworked to optimize the code and add additional features. The name was changed to IWBasic and, with few exceptions, is backward compatible to its compiler predecessors.

IWBasic and CBasic are both actively used and are supported via the Ionicwind forums.

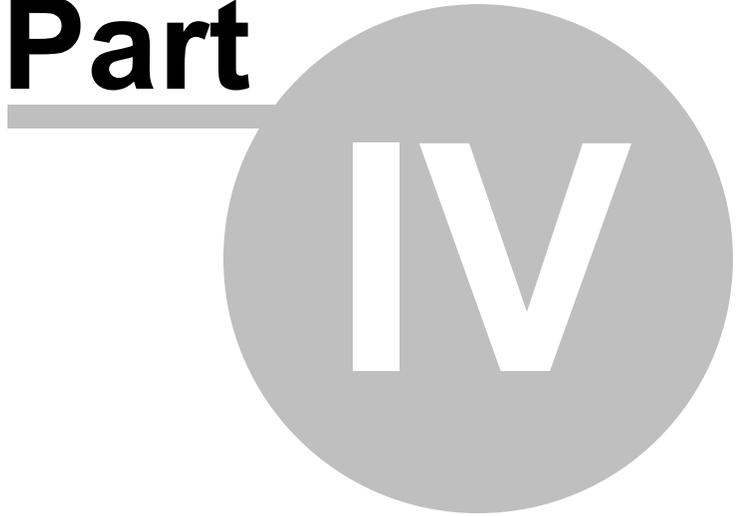
Upgrading

For Users upgrading from IBasic/CBasic, the [How-To»Convert from CBasic / IBasic](#) section covers most of the code changes required to run those programs in IWBasic.

For Users upgrading from IBPro/EBasic, the [How-To»Convert from EBasic / IBPro](#) section covers the few code changes required to run those programs in IWBasic.

IDE Interface

Part



4 IDE Interface

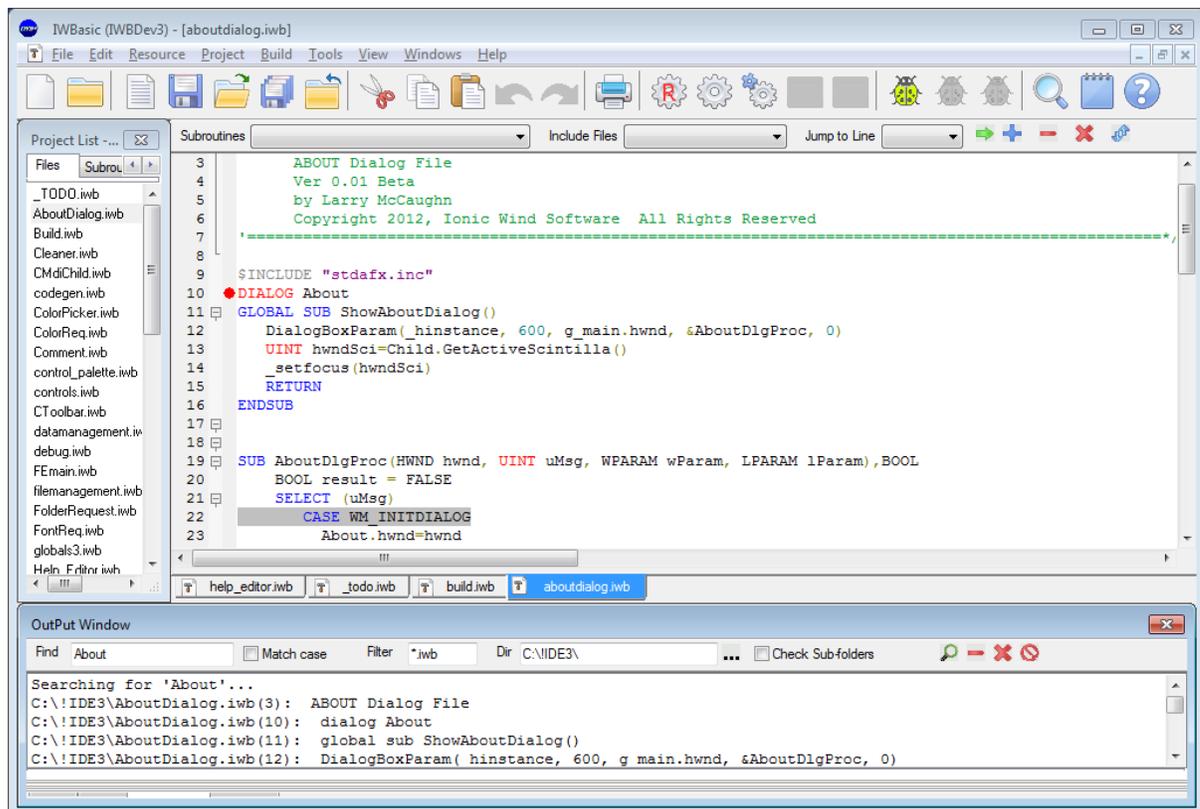
This section covers the IDE's User Interface.

4.1 Overview

IWBasic's IDE can be separated into seven principle sections .

The following screenshot is divided into those sections. Each section has its own page in the help file that describes the purpose and use of the section. Move the cursor over an area of the image to see the tooltip.

Click on an area to go to the page for that section.



4.2 Caption Bar

This section describes the Caption Bar area of IWBasic's IDE. Other portions of the help file may refer to this section when explaining an activity.



The Caption Bar always contains the IWBasic small icon and "IWBasic".

When a project is open the project name will be displayed. In the example above it is "(IWBDDev3)".

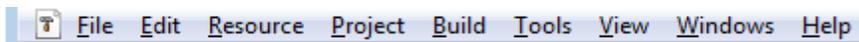
If a project has unsaved changes pending the project name will be followed by an asterisk "(IWBDev3*)".

When a source file is open and currently selected the file name will be displayed. In the example it is "[AboutDialog.iwb]".

If the currently selected file has unsaved changes pending the file name will be followed by an asterisk "[AboutDialog.iwb*]".

4.3 Main Menu

This section describes the Main Menu area of the IWBasic IDE. Other portions of the help file may refer to this section when explaining an activity.

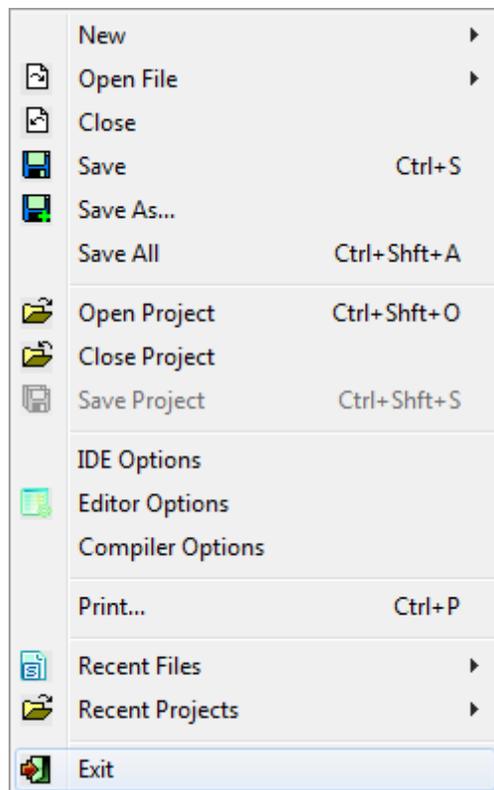


The Main Menu bar has 9 options: [File](#), [Edit](#), [Resource](#), [Project](#), [Build](#), [Tools](#), [View](#), [Windows](#), and [Help](#).

If there is a source file currently selected in the IDE the icon associated with that file type will appear at the left side of the Main Menu.

Each Main Menu option is covered in the sub-sections that follow.

File



Clicking on *File* results in the dropdown menu shown at left.

Each available option (and sub-option) is described below.

Option	Description
New	Opens a File / Project sub-menu (shown below).
Open File	Opens a Source / Example / Template sub-menu (shown below).
Close	Closes the currently selected file in the WorkSpace . If the current file has unsaved changes, the User will be prompted to save those changes before the file is closed. Disabled when there are no files open in the WorkSpace .
Save	Saves all changes made to the currently selected file in the WorkSpace . This option has a corresponding Main ToolBar button  . See the HowTo»Files»Save a File section for additional details. Disabled when there are no files open in the WorkSpace .
Save As...	Opens the 'Save File' dialog and allows the User to save the currently selected file in the WorkSpace to a new source file. Any unsaved changes to the currently selected file in the WorkSpace will NOT be saved to the currently selected file but will be saved to the new file. What occurs next depends on whether or not the currently selected file is a project or non-project file. See the How-To»Files»Save a File section for additional details. Disabled when there are no files open in the WorkSpace .
Save All	Saves all changes made to all currently open files in the WorkSpace . This option has a corresponding Main ToolBar button  . See the How-To»Files»Save a File section for additional details. Disabled when there are no files open in the WorkSpace .
Open Project	Opens the "Open File" dialog and allows the User to load an existing project. If there is a currently opened project it is closed. This option has a corresponding Main ToolBar button  . See the How-To»Projects»Open an Existing Project section for additional details.
Close Project	Closes the current project. If the current project has unsaved changes, the User will be prompted to save those changes before the project is closed. This option is disabled when no project is open. This option has a corresponding Main ToolBar button  . See the How-To»Projects»Close a Project section for additional details. Disabled when there is no open project.
Save Project	Saves all changes made to the current project excluding changes to source file edits. This option is disabled when there are no changes to save. See the How-To»Projects»Save a Project section for additional details.
IDE Options	Opens the "IDE General Preferences" dialog. See How-To»Set Startup Preferences section for additional details.
Editor Options	Opens the "Code Editor Preferences" dialog. See How-To»Set Editor Preferences section for additional details.

Compiler Options	Opens the "Compiler Properties" dialog. See How-To»Set Compiler Preferences section for additional details.
Print...	Opens a Print Preview dialog which facilitates the printing of source code. This option has a corresponding Main ToolBar button  . See The IDE»Features and Functions»Print Preview section for additional details. Disabled when there are no files open in the WorkSpace .
Recent Files	Clicking on Recent Files results in a popup menu (shown below) with a list of up to 30 of the last source files that were opened. The maximum number of files listed is User selectable. See How-To»Set Startup Preferences section for additional details.
Recent Projects	Clicking on Recent Projects results in a popup menu (shown below) with a list of up to 30 of the last projects that were opened. The maximum number of projects listed is User selectable. See How-To»Set Startup Preferences section for additional details.
Exit	If there is a current project, it is closed after prompting to save any unsaved changes. The User is prompted to save all open source files containing unsaved changes. Then IWBasic is closed.

New sub-menu

 File	Ctrl+N
 Project	Ctrl+Shft+N

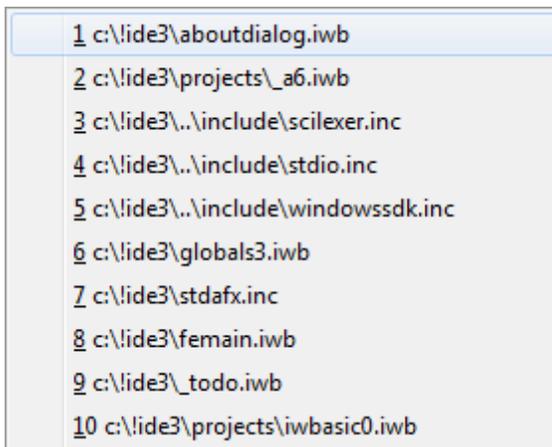
Option	Description
File	Creates a new IWBasic source file with an automatically generated filename in the WorkSpace . This option has a corresponding Main ToolBar button  . See the How-To»Files»Create a File section for additional details.
Project	Opens the "Create New Project" dialog This option has a corresponding Main ToolBar button  . See the How-To»Projects»Create a New Project section for additional details.

Open File sub-menu

 Source	Ctrl+O
 Example	Ctrl+Shft+E
 Template	Ctrl+Shft+T

Option	Description
Source	Opens the "Open File" dialog and allows the User to load an existing file. This option has a corresponding Main ToolBar button  . The default directory is the last directory used to open a file. See the How-To»Files»Open a File section for additional details.
Example	Opens the "Open File" dialog and allows the User to load an existing file. The default directory is the directory used to store example files. See the How-To»Files»Open a File section for additional details.
Template	Opens the "Open File" dialog and allows the User to load an existing file. The default directory is the directory used to store template files. See the How-To»Files»Open a File section for additional details.

Recent Files sub-menu

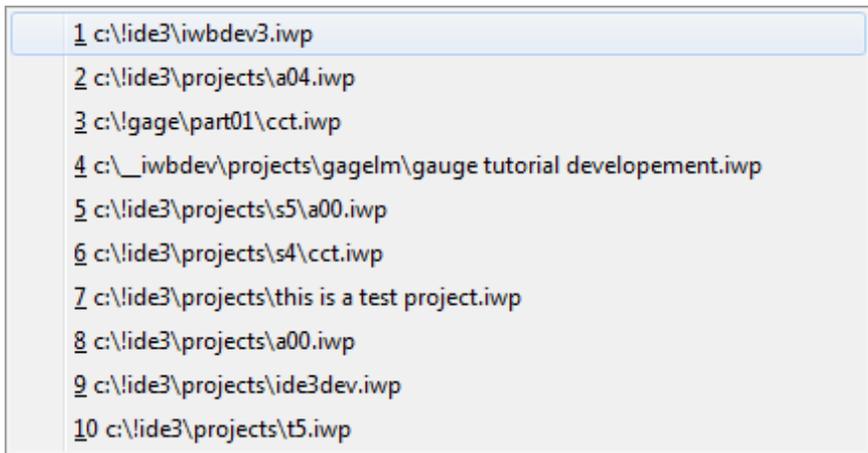


An example of the *Recent Files* sub-menu.

The files will be listed with the most recent first and the least recent last.

Clicking an entry will open the selected file in the [WorkSpace](#). If the selected file no longer exists the User will be notified, the entry removed, and the popup redisplayed.

Recent Projects sub-menu



An example of the *Recent Projects* sub-menu.

The projects will be listed with the most recent first and the least recent last.

Clicking an entry will open the selected project. If the selected project no longer exists the User will be notified, the entry removed, and the popup redisplayed. If a valid selection is made the User is prompted to save any unsaved changes to the current project, if one exists.

Edit

Undo	Ctrl+Z
Redo	Ctrl+Y
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Select All	Ctrl+A
Find	Ctrl+F
Replace	Ctrl+H

Clicking on *Edit* results in the dropdown menu shown at left. Each available option is described below.

Option	Description
Undo	Used to undo an edit. Disabled unless there is an unsaved change in the currently selected source file in the WorkSpace . This option has a corresponding Main ToolBar button  . See Utilities»Code Editor»Editing Text for additional information.
Redo	Used to redo an edit that has been undone. Disabled unless there is a previous 'undo' action in the currently selected source file in the WorkSpace . This option has a corresponding Main ToolBar button  . See Utilities»Code Editor»Editing Text for additional information.
Cut	Used to delete selected text from the currently selected source file in the WorkSpace . Disabled when there is no text selected. This option has a corresponding Main ToolBar button  . See Utilities»Code Editor»Editing Text

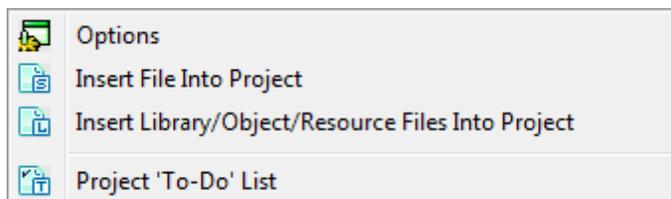
	for additional information.
Copy	Used to copy selected text in the currently selected source file in the WorkSpace to the clipboard. Disabled when there is no text selected. This option has a corresponding Main ToolBar button  . See Utilities»Code Editor»Editing Text for additional information.
Paste	Used to paste/insert the contents of the clipboard in place of the currently selected text or at the current caret location in the currently selected source file in the WorkSpace . Disabled when clipboard is empty. This option has a corresponding Main ToolBar button  . See Utilities»Code Editor»Editing Text for additional information.
Select All	Used to select all the text in the currently selected source file in the WorkSpace . Disabled when there is no file currently selected. See Utilities»Code Editor»Editing Text for additional information.
Find	Used to find text in the currently selected source file in the WorkSpace . Opens a "Find" dialog. See How-To»Files»Find Text in a File for additional information.
Replace	Used to replace text in the currently selected source file in the WorkSpace . Opens a "Replace" dialog. See How-To»Files»Replace Text in a File for additional information.

Resource



Option	Description
Add	Used to open the Output Window (if it is not currently open) and selects the Resources tab. From there the User may add resources to the current project. Disabled when no project is currently open. See How-To»Projects»Add a Resource for additional information.

Project

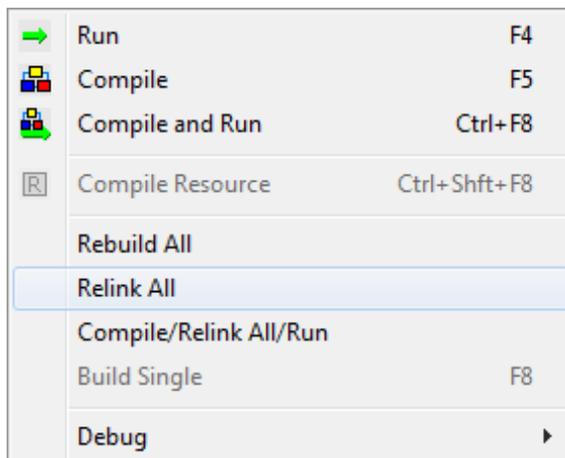


Clicking on *Project* results in the dropdown menu shown at left. Each available option is described below.

Option	Description
--------	-------------

Options	Opens the "Modify Project Options" dialog, which allows the User to modify a limited number of options for the current project. See the How-To »Projects»Set Project Options section for additional details.
Insert File Into Project	Inserts the currently selected non-project source file into the current project. Disabled when there is no current project. Disabled when the currently selected source file is not a non-project file. See the How-To»Projects»Add Files section for additional details.
Insert Library /Object / Resource Files Into Project	Opens the "Add obj, res, or lib File" dialog which allows the User to select one or more of the indicated file types into the project. See the How-To»Projects»Add Files section for additional details.
Project 'To-Do' List	Opens the "Project ToDo List" dialog which allows the User to maintain a list of things to do for the current project. See the To-Do List section for additional details.

Build



Clicking on *Build* results in the dropdown menu shown at left with the following options (described below).

Option	Description
Run	Executes the exe file for the current project, if it exists. If there is no current project then the current single file application exe file will be executed, if it exists. Disabled if there is no current exe file. See the How-To »Projects»Run a Project section and How-To »Single File Application»Run an Application section for additional details. This option has a corresponding Main ToolBar button  .

Compile	Compiles the currently selected source file of the current project. Disabled if there is no current project. See the How-To» Projects»Compile a Project section for additional details. This option has a corresponding Main ToolBar button  that works for both projects and single file applications. Note: The  button compiles all source files and the resource file for projects.
Compile and Run	Compiles the current project and, if successful, executes the resulting exe file. If there is no current project but is a current Single File application then the current single file app is compiled and, if successful, executes the resulting exe file.. See the How-To» Projects»Compile a Project and How-To» Projects»Run a Project or the How-To»Single File Application»Compile an Application and How-To»Single File Application»Run an Application sections for additional details. This option has a corresponding Main ToolBar button  that works for both projects and single file applications.
Compile Resource	Compiles the current project's resource file. See the How-To» Projects»Compile a Resource section for additional details. This option has a corresponding Main ToolBar button  . Disabled if there is no current project.
Rebuild All	Compiles all of the source files of the current project. If successful the project is then relinked into an executable. Disabled if there is no current project. See the How-To» Projects»Compile a Project section for additional details.
Relink All	Relinks all of the files of the current project into an executable. Disabled if there is no current project. See the How-To» Projects»Compile a Project section for additional details.
Compile/ Relink All/ Run	Compiles the currently selected source file of the current project. Disabled if there is no current project. If the compile is successful then the object file will be re-linked with all the other object files in the project. If the re-link is successful then the project will be executed.
Build Single	Compiles the currently selected source file. Disabled if there is a current project. See the How-To »Single File Application»Compile an Application section for additional details.
Debug	Opens a Debug sub-menu (shown below).

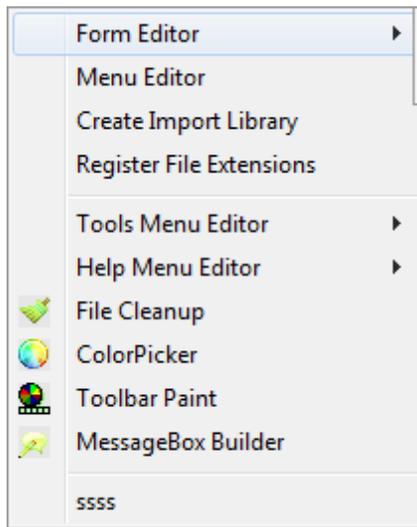
Debug sub-menu

	Start	F9
	Single Step	F10
	Stop	F7

Option	Description
--------	-------------

Start	Start/Continue a debug session. See the Debugging Programs section for additional details.. This option has a corresponding Main ToolBar button  .
Single Step	Single Step through a program during Debug. See the Debugging Programs section for additional details.. This option has a corresponding Main ToolBar button  .
Stop	Stop a Debug session. See the Debugging Programs section for additional details.. This option has a corresponding Main ToolBar button  .

Tools

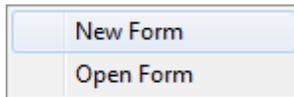


Clicking on *Tools* results in the dropdown menu shown at left with the following options (described below).

Option	Description
Form Editor	Opens a New Form/ Open Form sub-menu (shown below). Disabled when the Form editor is already open. See the Utilities»Form Editor section for additional details.
Menu Editor	Opens the Menu Editor utility for creating application menus. See the Utilities»Menu Editor section for additional details.
Create Import Library	Opens the "Select DLL" dialog for creating Import libraries. See See the Utilities»Create Import Library section for additional details.
Register File Extentions	Opens the Register File Extension utility for establishing IWBasic as the preferred program for opening various files. See the Utilities»Register File Extension section for additional details.
Tools Menu Editor	Opens a File / Project sub-menu (shown below).

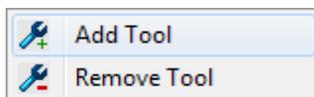
Help Menu Editor	Opens a File / Project sub-menu (shown below).
File Cleanup	Opens the File Cleanup utility for removing temporary files associated with IWBasic.. See the Utilities»File Cleanup section for additional details.
ColorPicker	Opens the ColorPicker utility for selecting and previewing color selections for an application's GUI. See the Utilities»ColorPicker section for additional details.
Toolbar Paint	Opens the Toolbar Paint utility for creating application toolbar bitmaps. See the Utilities»Toolbar Paint section for additional details.
MessageBox Builder	Opens the MessageBox Builder utility for creating various application Messageboxes. See the Utilities»MessageBox Builder section for additional details.
...	Following all the above menu options will be those utilities add by the User via the Tool Menu Editor. See the Utilities»Tool Menu Editor section for additional details.

Form Editor sub-menu



Option	Description
New Form	Opens the "New Form" dialog and allows the User to create a new form file. See the Utilities»Form Editor section for additional details.
Open Form	Opens the "Load Form File" dialog and allows the User to select an existing form file. The Form Editor is opened and the selected file is loaded for editing. See the Utilities»Form Editor section for additional details.

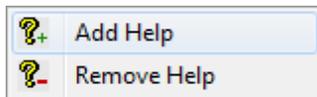
Tools Menu Editor sub-menu



Option	Description
Add Tool	Opens the "Add Tool" dialog, which allows the User to add additional tools to the

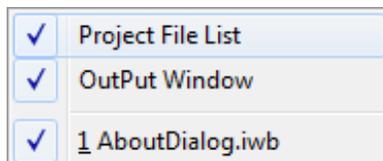
	Tools menu. See the Utilities»Tools Menu Editor section for additional details.
Remove Tool	Opens the "Remove Tool" dialog, which allows the User to remove items from the Tools menu that were added by the User. See the Utilities»Tools Menu Editor section for additional details.

Help Menu Editor sub-menu



Option	Description
Add Help	Opens the "Add Tool" dialog, which allows the User to add additional tools to the Tools menu. See the Utilities»Help Menu Editor section for additional details.
Remove Help	Opens the "Remove Tool" dialog, which allows the User to remove items from the Tools menu that were added by the User. See the Utilities»Help Menu Editor section for additional details.

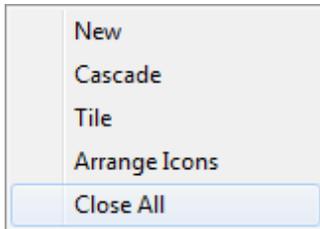
View



Clicking on *View* results in the dropdown menu shown at left with the following options (described below).

Option	Description
Project File List	Toggles the current visible state of the Project List Window. See the Project List Window section for additional details.
Output Window	Toggles the current visible state of the Output Window. See the Output Window section for additional details.
...	Following the above menu options will be a list of all the files currently open in the Workspace . The currently selected file will have a checkmark. The User may select a file by clicking the desired filename.

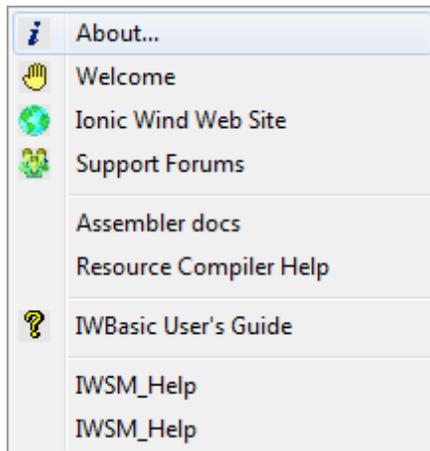
Windows



Clicking on *Windows* results in the dropdown menu shown at left with the following options (described below).

Option	Description
New	Creates a new IWBASIC source file with an automatically generated filename in the WorkSpace . This option has a corresponding Main ToolBar button  and a Main Menu <i>File/New/File</i> option. See the How-To»Files»Create a File section for additional details.
Cascade	Automatically resizes all open windows in the WorkSpace and arranges them in an overlapping manner
Tile	Automatically resizes all open windows in the WorkSpace and arranges them in a tiled manner
Arrange Icons	Automatically arranges the icons for all the minimized windows in the WorkSpace .
Close All	Closes all the files currently open in the WorkSpace . The User will be prompted to save any files with pending un-saved changes.

Help



Clicking on *Help* results in the dropdown menu shown at left with the following options (described below).

Option	Description
About	Opens the "About" dialog which shows the User the current version info for IWBasic.
Welcome	Displays a web browser window with some useful information and links.
Ionic Wind Web Site	Opens the User's web browser to the Ionic Wind web site main page.
Support Forums	Opens the User's web browser to the Ionic Wind support forums main web page.
Assembler docs	Opens the Help file associated with the NASM assembler.
Resource Compiler Help	Opens the Help file associated with the GORC resource compiler.
IWBasic User's Guide	Opens this help file.
...	Following all the above menu options will be those Help files added by the User via the Help Menu Editor.. See the Utilities»Help Menu Editor section for additional details.

4.4 Main ToolBar

This section describes the *Main ToolBar* area of IWBASIC. Other portions of the help file may refer to this section when explaining an activity.



The main tool bar has 24 buttons. The function of each is described below.

Clicking on a toolbar button results in the described action:

Button	Description
	Creates a new, automatically named, IWBASIC source file, and opens it in the WorkSpace . This option has a corresponding Main Menu File option. See the How-To» File» Create a New File section for additional details.
	Opens the "New Project" dialog, creates a new IWBASIC project, and opens it. If there is a project currently opened, it will be closed. If the current project has unsaved changes, the User will be prompted to save those changes before the project is closed. This option has a corresponding Main Menu File/New/Project option. See the How-To» Projects» Create a New Project section for additional details.
	Opens the "Open File" dialog and allows the User to load an existing file in the WorkSpace . This option has a corresponding Main Menu File/Open File option. See the How-To»File»Open an Existing File section for additional details.
	Saves all changes made to the current select file. This option is disabled when there are no changes to save. This option has a corresponding Main Menu File/Save option. See the How-To»File»Save a File section for additional details.
	Opens the "Open Project" dialog and allows the User to load an existing project. If there is a currently opened project it is handled as described above. This option has a corresponding Main Menu File/Open Project option. See the How-To»Projects»Open an Existing Project section for additional details.
	Saves all changes made to the files open in the WorkSpace . This option is disabled when there are no changes to save. This option has a corresponding Main Menu File/Save All option. See the How-To» File»Save a File section for additional details.

	Closes the current project. If the current project has unsaved changes, the User will be prompted to save those changes before the project is closed. This option has a corresponding Main Menu File/Close Project option. See the How-To»Projects»Close a Project section for additional details.
	Used to delete selected text from the currently selected source file in the WorkSpace . Disabled when there is no text selected. This option has a corresponding Main ToolBar button  . See Utilities»Code Editor»Editing Text for additional information.
	Used to copy selected text in the currently selected source file in the WorkSpace to the clipboard. Disabled when there is no text selected. This option has a corresponding Main ToolBar button  . See Utilities»Code Editor»Editing Text for additional information.
	Used to paste/insert the contents of the clipboard in place of the currently selected text or at the current caret location in the currently selected source file in the WorkSpace . Disabled when clipboard is empty. This option has a corresponding Main ToolBar button  . See Utilities»Code Editor»Editing Text for additional information.
	Used to undo an edit. Disabled unless there is an unsaved change in the currently selected source file in the WorkSpace . This option has a corresponding Main ToolBar button  . See Utilities»Code Editor»Editing Text for additional information.
	Used to redo an edit that has been undone. Disabled unless there is a previous 'undo' action in the currently selected source file in the WorkSpace . This option has a corresponding Main ToolBar button  . See Utilities»Code Editor»Editing Text for additional information.
	Opens a Print Preview dialog which facilitates the printing of source code. This option has a corresponding Main ToolBar button  . See Utilities»Print Preview section for additional details. Disabled when there are no files open in the WorkSpace .
	Compiles the current project's resource file. This option has a corresponding Main Menu Build option. See the How-To»Projects»Compile a Resource section for additional details.
	Compiles the current project or single file application. This option has a corresponding Main Menu Build option for projects. See the How-To»Projects»Compile a Project or How-To»Single File Application»Compile an Application sections for additional details.
	Compiles the current project or single file application and, if successful, executes the resulting exe file. This option has a corresponding Main Menu Build option for

	projects. See the How-To»Projects»Run a Project and How-To»Projects»Compile a Project sections or the How-To»Single File Application»Compile an Application and How-To»Single File Application»Run an Application sections for additional details.
	Executes the exe file for the current project or single file application, if it exists. This option has a corresponding Main Menu Build option for projects. See the How-To»Projects»Run a Project or How-To»Single File Application»Run an Application sections for additional details.
	Opens the "Executable Options" dialog for single file apps. See the How-To»Single File Application»Set Application Options section for additional details. Disabled when a project is open and when a file is not currently selected in the WorkSpace .
	Start/Continue DEBUG mode. See the IDE Modes»Debugging Programs section for additional details.
	Single-Step in DEBUG mode. See the IDE Modes»Debugging Programs section for additional details.
	Stop DEBUG mode. See the IDE Modes»Debugging Programs section for additional details.
	Selects the "Find in Files" tab of the Output Window . Opens the Output Window if it is not open. See the Utilities»Find in Files section for additional details.
	Opens the "Project To-Do List" for the current project if one is open. Otherwise, the global "IDE To-Do List" is opened. See the To-Do List section for additional details.
	Opens this help file. This option has a corresponding Main Menu Help/IWBasic User's Guide option.

4.5 WorkSpace

The *WorkSpace* is the heart of the IDE. All actual programming takes place in this area.

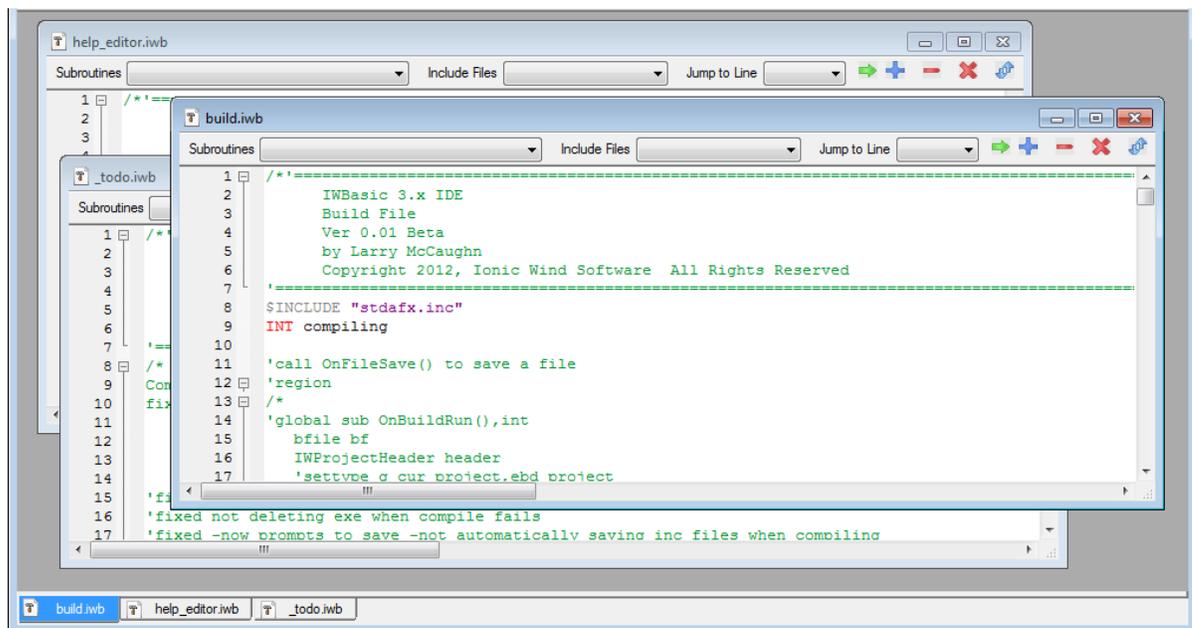
The main portion of the *WorkSpace* contains multiple instances of the [Code Editor](#). Each instance contains the contents of a single file.

Along the top edge of the *WorkSpace* is a tab control that will contain one tab for each [Code Editor](#) instance. Each tab will contain the file's name.

Clicking on a tab will bring that file to the forefront and make it the currently selected file. Likewise, clicking on any portion of a [Code Editor](#) instance will cause that instance to become the currently selected file and will cause the tab control to automatically adjust to reflect the selection.

The number of files open in the *WorkSpace* is limited only by available memory.

See the [How-To»Files](#) section for details about creating, opening, saving, and closing files. See the [Utilities»Code Editor](#) section for details about the functionality of each instance of a [Code Editor](#) window.



4.6 Project List Window

Introduction

The *Project List* window contains information specific to the current project if one is open.

The *Project List* can be opened/closed via the [View / Project File List](#) option on the [Main Menu](#).

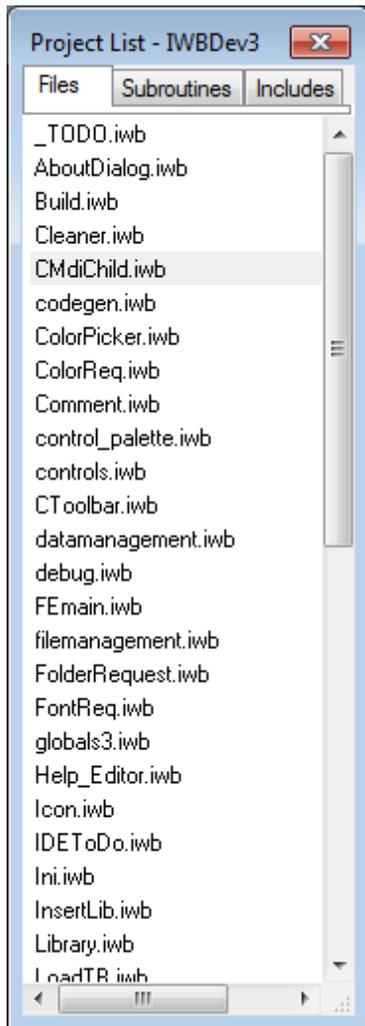
The window can also be closed via the window's close button.

The *Project List's* caption contains the name of the current project.

The height of the *Project List* automatically tracks the height of the main IDE window. The width of the *Project List* can be increased/decreased by dragging the edge not touching the main IDE window.

The *Project List* contains three tabs, [Files](#), [Subroutines](#), [Includes](#)) that control which list is currently being displayed.

Files



The *Files* tab contains the name of every source file that has been added to the project. See the [How-To»Projects»Add Files](#) section for additional details on adding files.

Double-clicking a file name will open the selected file in a new instance of the [Code Editor](#) in the [WorkSpace](#), ready for editing. Note: Binary files will not be opened.

Right-clicking in this tab opens the following popup menu.

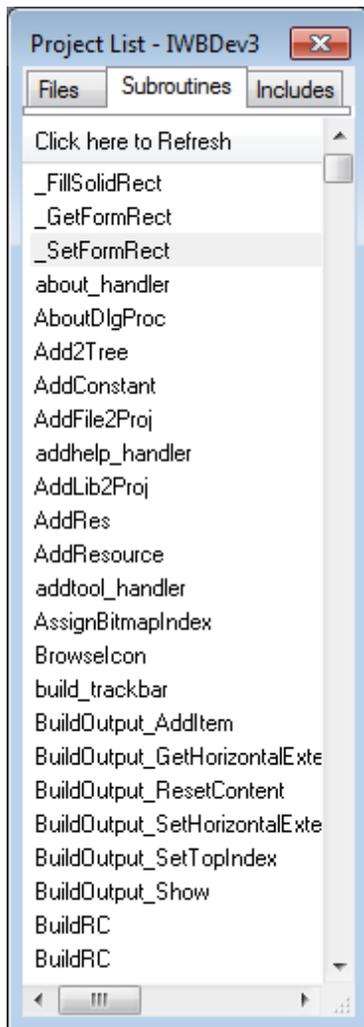
Remove File from Project

The User is presented with one option.

- *Remove File from Project* - will remove the currently selected file from the project. The User will be prompted to confirm the removal. Although the file will be removed it will not be deleted..

The current location of the *Project List* will be indicated in the menu via a check mark.

Subroutines



The *Subroutines* tab contains the name of every subroutine in every source file in the project. The list includes class methods also.

Double-clicking a subroutine name will open the parent file in a new instance of the [Code Editor](#) in the [WorkSpace](#), ready for editing. The caret in the [Code Editor](#) will be positioned to the start of the subroutine.

At the top of the list is a *Click here to Refresh* button. This allows the User to insure that the list reflects any recent additions/deletions/renaming.

The current location of the *Project List* will be indicated in the menu via a check mark.

Includes



The *Includes* tab contains the name of every file in the project that is included via the \$INCLUDE directive. The list does not include nested files with the exception of the stdafx.inc file.

Double-clicking a file name will open the file in a new instance of the [Code Editor](#), ready for editing.

At the top of the list is a *Click here to Refresh* button. This allows the User to insure that the list reflects any recent additions/deletions/renaming.

The current location of the *Project List* will be indicated in the menu via a check mark.

4.7 Output Window

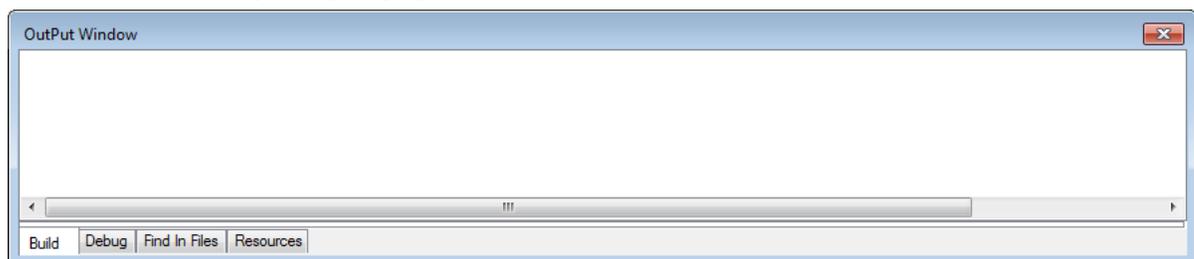
Introduction

The *Output Window* window (shown below) is a general purpose window for displaying a variety of data.

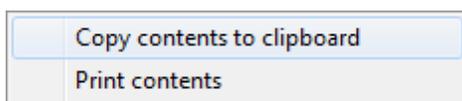
The *Output Window* can be opened/closed via the [View/Output](#) option on the [Main Menu](#). The window can also be closed via the window's close button.

The width of the *Output Window* automatically tracks the width of the main IDE window. The height of the *Output Window* can be increased/decreased by dragging the edge not touching the main IDE window.

The *Output Window* contains four tabs, [Build](#), [Debug](#), [Find in Files](#), [Resources](#), that control what information is currently being displayed.



Right clicking in the window results in the following popup menu for all tabs except Resources.



Each available option is described below.

The User is presented with two options.

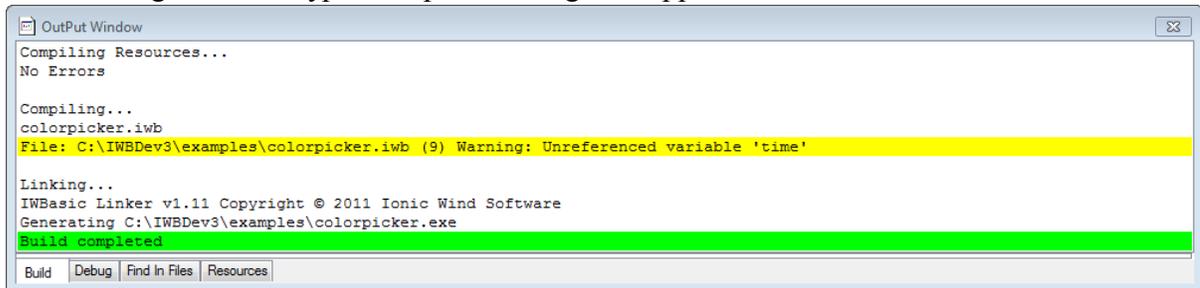
- *Copy contents to clipboard* - Copies the entire contents currently displayed in *Output Window* to the clipboard.

- *Print contents* - Copies the entire contents currently displayed in *Output Window* to the printer.

Build

The *Build* tab is used to display the progress of any compile or link activity. If the *Output Window* is not currently open when a compile/link activity is initiated it will be automatically opened and the *Build* tab automatically selected. If the *Output Window* is open but the *Build* tab is not currently selected the *Build* tab will be automatically selected.

The following shows the typical output for a single file application build.



```
OutPut Window
Compiling Resources...
No Errors

Compiling...
colorpicker.iwb
File: C:\IWBDev3\examples\colorpicker.iwb (9) Warning: Unreferenced variable 'time'

Linking...
IWBASIC Linker v1.11 Copyright © 2011 Ionic Wind Software
Generating C:\IWBDev3\examples\colorpicker.exe
Build completed

Build | Debug | Find In Files | Resources
```

If an error or warning condition is encountered during the compile process a line of text is generated and displayed. The line will contain the full path name of the source file, the line number of the line in the source file where the problem occurred, and the nature of the problem.

If the problem is just a warning the problem information line will be colored yellow as shown above. The compile/link process will continue. If the problem is an error the information line will be colored red. The final link will fail if this happens.

Double-clicking the red or yellow informational line will automatically open the indicated source file (if it is not already open) in a new [Code Editor](#) instance in the [WorkSpace](#). The [Code Editor](#) window will automatically be scrolled to the indicated line number.

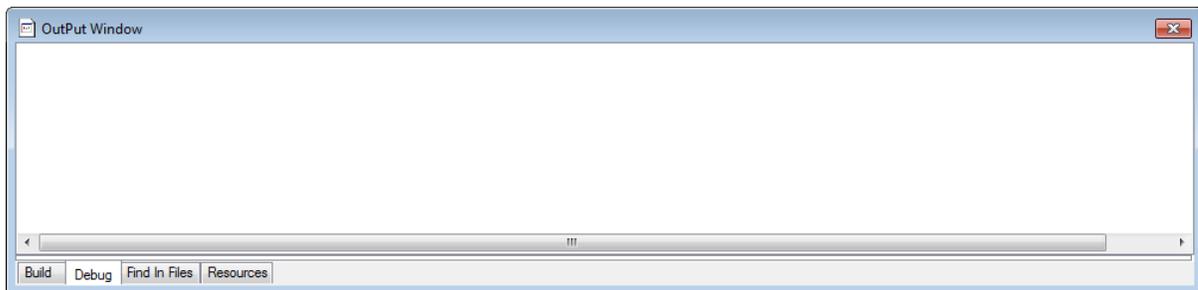
If the build is successful the final line will be colored green as shown above. If the build fails for any reason the final line will be colored red.

As indicated previously the User can right-click and copy the contents to the clipboard. This will allow the User to easily share accurate information when seeking help from others.

Debug

The *Debug* tab is used to display the progress of any debug activity. If the *Output Window* is not currently open when a debug activity is initiated it will be automatically opened and the *Debug* tab automatically selected. If the *Output Window* is open but the *Debug* tab is not currently selected the *Debug* tab will be automatically selected.

The following shows the typical output for a debug event. See the [IDE Modes»Debugging programs](#) section for additional information.



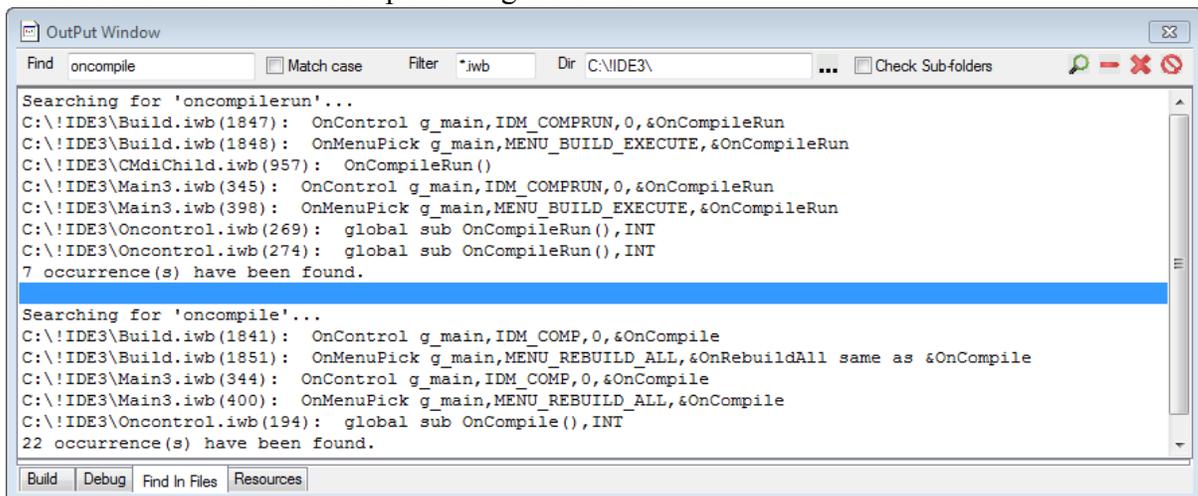
As indicated previously the User can right-click and copy the contents to the clipboard. This will allow the User to easily share accurate information when seeking help from others.

Find in Files

The *Find in Files* tab is used to display the results of one or more searches. When the User clicks the *Find in Files* button  on the [Main Toolbar](#), the *Output Window* will open if it is not currently open and the *Find in Files* tab will be automatically selected.

Double-clicking a result line will automatically open the indicated source file (if it is not already open) in a new [Code Editor](#) instance in the [WorkSpace](#). The [Code Editor](#) window will automatically be scrolled to the indicated line number.

The following shows the typical results for multiple searches. See the [Utilities»Find in Files](#) section for additional information about performing searches.



Resources

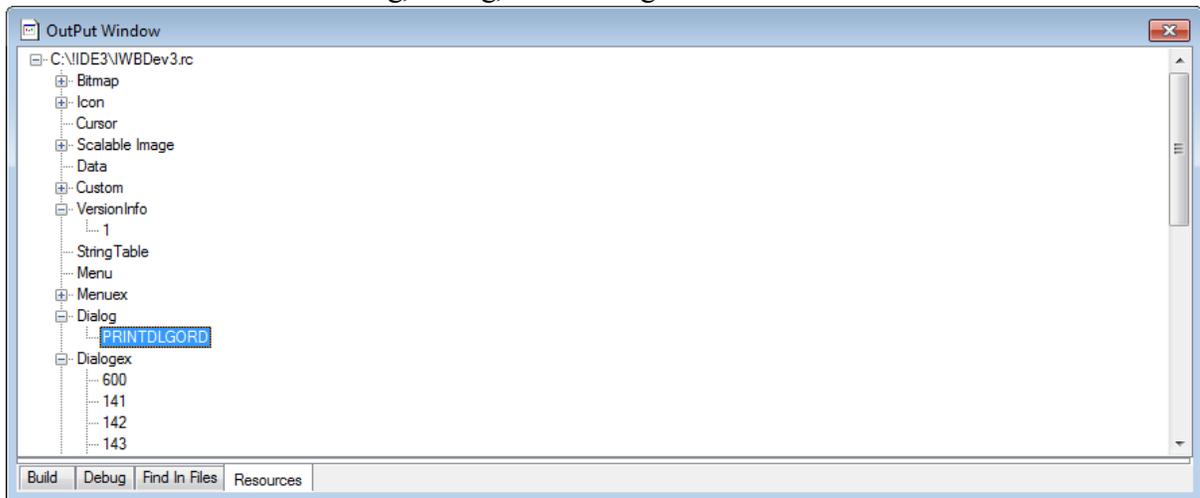
The *Resources* tab contains all the resources that have been added to a project. The window will be empty when there is no currently loaded project. When the User clicks the [Main Menu Resources»Add](#) option is selected the *Output Window* will open if it is not currently open and the *Resources* tab will be automatically selected.

The *Resources* tab contains a tree-view with each of the following pre-defined resource types:

- Bitmap
- Icon
- Cursor

- Scalable Image
- Data
- Custom
- VersionInfo
- StringTable
- Menu
- Menuex
- Dialog
- Dialogex
- Manifest

The following shows a *Resources* tab. See the [How-To»Projects»Add Files»Resources](#) for detailed information about adding, editing, and deleting resources.



Right clicking in the window results in one of the two following popup menus when the *Resources* tab is selected..

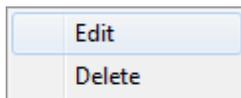
If the mouse cursor is on a line containing a resource type the following menu will appear.



Opens the proper "Add" dialog for the selected resource type.

See the [How-To»Projects»Add Files»Resources](#) section for detailed information.

If the mouse cursor is on a line containing the ID of an existing resource the following menu will appear.



Each available option is described below.

The User is presented with four options.

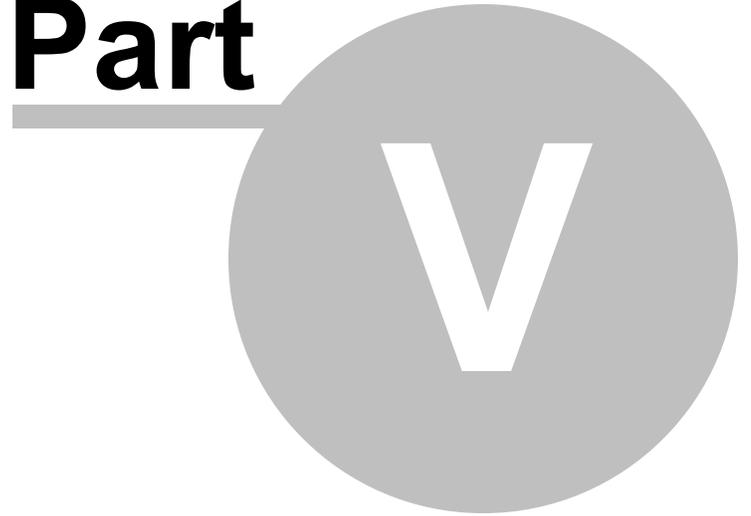
- *Edit* - Opens an "Edit" dialog pre-loaded with the selected resource's information..

- *Delete* - Deletes the selected resource entry after confirmation.

See the [How-To»Projects»Add Files»Resources](#) section for detailed information.

IDE Modes

Part



5 IDE Modes

The IWBASIC IDE is dual mode in that it allows compiling a single file directly to an executable or DLL; or combining multiple source files and resource information into a centralized project module which is then compiled to an executable, DLL, or static library .

- [Single File Programs](#)
- [Multi File Programs](#)

5.1 Single file programs

Introduction

While projects may be useful for larger programs they are a bit cumbersome if your program only consists of one source file and you want to quickly make and test changes to your code. Most of the examples provided with IWBASIC are single file programs which can be compiled directly to an executable or DLL.

Steps to use single file mode

1. Make sure there isn't a project open by selecting the [Main Menu File/Close Project](#) option or clicking the [Main ToolBar](#)  button.
2. Open an existing IWBASIC source file by one of the methods described in [How-To»Files»Open a File](#). Or create a new source file by one of the methods described in [How-To»Files»Create a File](#).
3. Select the [Main Menu Build/Build Single](#) or click the [Main ToolBar](#)  button. This will display the *Executable Options* dialog. Note: If you created a new source file you will be prompted to save first before the dialog is opened.
4. Set the desired options in the [Executable Options](#) dialog and click the [Executable Options](#) dialog's *Create* button. See the [How-To»Single File Application](#) section for additional details about [Executable Options](#) dialog options.
5. Observe any compiler, assembler or linker error messages in the [Output Window](#) of the IDE. Correct any errors and recompile as necessary. As in project mode you can double click on any error occurring in an IWBASIC source file to highlight the offending line. See the [Output Window](#) section for additional details.

Executing your program

After a successful build of the program you can test the resulting executable, if any, by one of the methods described in the [How-To»Single File Application»Run an Application](#) section.

Notes

Any options selected in the [Executable Options](#) dialog will be remembered for the next compile. The options are stored in a separate file with the source file name and a ".opts" extension in the

same directory as the source file. If this file is deleted, or doesn't yet exist, the [Executable Options](#) dialog will display default values.

The only resource available for use in single file mode is the program's icon selected with the [Executable Options](#) dialog.

It is not necessary to use *\$main* in single file mode. The compiler automatically determines the point of execution as there is only one file. The *\$main* keyword will be silently ignored.

Be sure to choose the proper output type. If you select a *Console* type for a *Windows* program you will get an error message about a missing `_window_list` variable from the linker. This error is normal and means you need to change the output type to *Windows*.

You can still use the [\\$include](#) command in your single file. Use this to bring in additional source files.

See Also: [Multi-file programs](#)

5.2 Multi-file programs

Introduction

Projects are the heart of multi module programming and combine all of the source files and resource information into a centralized project module. Using projects you can create very complex programs by separating your source code over multiple files and libraries. Subroutines and variables in source files can be either local or shared globally between files for ease of data reuse.

Steps to use multi file mode

1. Create a new project by selecting the [Main Menu File/New/Project](#) option or click the [Main ToolBar](#)  button to open the "Create New Project" dialog. Complete all the entries in the "Create New Project" dialog as described in the [How-To»Projects»Set Project Options](#) section.
2. Or, select an existing project by selecting the [Main Menu File/ Open Project](#) option or click the [Main ToolBar](#)  button to open the "Load Project" dialog. Select the desired *.iwp/*.ebp project file to open. See the [How-To»Projects»Open a Project](#) section for additional information.
3. Optionally, open the *Project List* window via the [Main Menu View / Project File List](#) menu option. See the [IDE Interface»Project List Window](#) for additional details.
4. Add source files to the project as described in the [How-To»Projects»Add Files](#) section. See note below for *\$main* directive requirement.
5. For existing projects remove any source files that are no longer required. See the [How-To»Projects»Remove Files](#) section for details.
6. Add any required project resources. See the [How-To»Projects»Resources](#) section for details.
7. Optionally, open the [Modify Project](#) dialog via the [Main Menu Project / Options](#) menu option. Change any desired options. See the [How-To»Projects»Set Project Options](#) section for

additional details.

8. Compile the project by selecting the [Main Menu Build/Rebuild All](#) or click the [Main ToolBar](#)  button
9. Observe any compiler, assembler or linker error messages in the [Output Window](#) of the IDE. Correct any errors and recompile as necessary. As in single-file mode you can double click on any error occurring in an IWBASIC source file to highlight the offending line. See the [Output Window](#) section for additional details.

Executing your program

After a successful build of the program you can test the resulting executable, if any, by one of the methods described in the [How-To»Projects»Run a Project](#) section.

The *\$main* directive

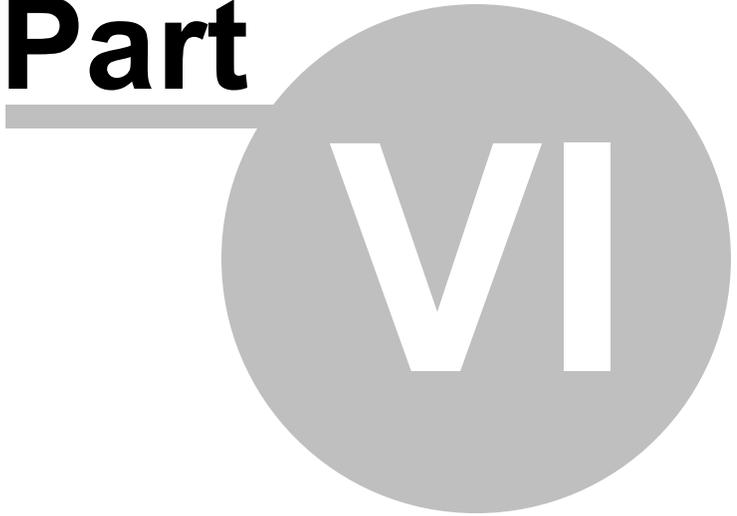
When using projects it is necessary to tell the compiler where to start execution of your program. This is accomplished by using the preprocessor keyword *\$main*. Only one source file can contain the *\$main* keyword and it does not matter where the keyword appears. Execution will begin at the first statement or command that is not in a function. DLL's and Compile-Only projects do not require the *\$main* directive

See Also: [Using Resources](#)

See Also: [Single file compiling](#)

Utilities

Part

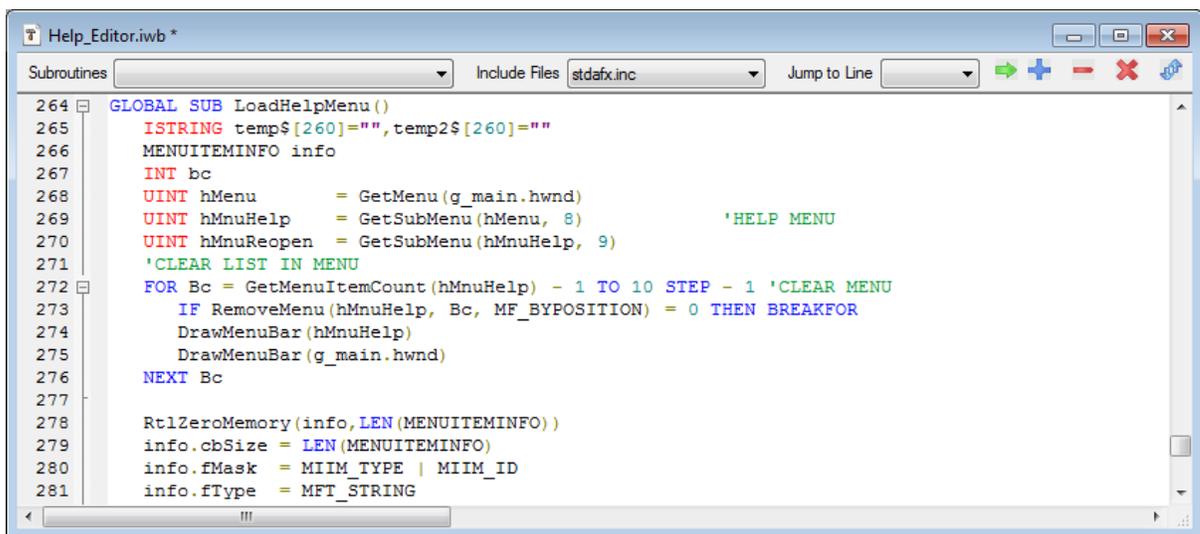


6 Utilities

6.1 Code Editor

Introduction

When the User creates a new ([How-To»Files»Create a File](#)) or opens an existing ([How-To»Files»Open a File](#)) source file, the IDE creates a new window in the [WorkSpace](#) that contains a single, custom, Scintilla-based text editor. An example of an existing file opened in an instance of the *Code Editor* is shown below.



```

Help_Editor.iwb *
Subroutines
Include Files stdafx.inc
Jump to Line
264 GLOBAL SUB LoadHelpMenu()
265 ISTRING temp${260}="", temp2${260}=""
266 MENUITEMINFO info
267 INT bc
268 UINT hMenu = GetMenu(g_main.hwnd)
269 UINT hMnuHelp = GetSubMenu(hMenu, 8) 'HELP MENU
270 UINT hMnuReopen = GetSubMenu(hMnuHelp, 9)
271 'CLEAR LIST IN MENU
272 FOR Bc = GetMenuItemCount(hMnuHelp) - 1 TO 10 STEP - 1 'CLEAR MENU
273 IF RemoveMenu(hMnuHelp, Bc, MF_BYPOSITION) = 0 THEN BREAKFOR
274 DrawMenuBar(hMnuHelp)
275 DrawMenuBar(g_main.hwnd)
276 NEXT Bc
277
278 RtlZeroMemory(info, LEN(MENUITEMINFO))
279 info.cbSize = LEN(MENUITEMINFO)
280 info.fMask = MIIM_TYPE | MIIM_ID
281 info.fType = MFT_STRING
  
```

Each *Code Editor* window consists of three parts; a [Work Area](#), [Title Bar](#), and [Toolbar](#).

Work Area

The *Code Editor* has many of the common features of a standard text editor plus several enhanced features.

The following are global aspects of the *Code Editor* that are User configurable. See the [How-To»Set Editor Preferences](#) for details.

- Font Name
- Font Size
- Tab key spacing
- General text color
- Window background color
- Keyword color
- Comment color
- Constant color
- Definition color
- Numeric color
- String color

- Operator color
- Assembly color
- Preprocessor color
- AutoTip - on/off (See the [AutoTip](#) section below for details)
- AutoTip highlight color
- AutoTip text color
- AutoTip background color
- Line numbers on/off
- Auto Indent on/off
- Autocomplete on/off (See the [Autocomplete](#) section below for details)
- Key word format

The following table identifies some of the key commands the editor responds to.

Key	Action
<Page Up>	Moves one page up
<Page Down>	Moves one page down
Up Arrow	Moves one line up
Down Arrow	Moves one line down
<CTRL> + A	Selects all text
<CTRL> + C	Copies selected text to clipboard
<CTRL> + F	Opens the 'Find' dialog
<CTRL> + H	Opens the 'Find/Replace' dialog
<CTRL> + X	Cuts selected text and copies to clipboard
<CTRL> + V	Pastes text from clipboard at caret position
<CTRL> + Z	Undoes the previous text operation (Undo)
<CTRL> + Y	Reverses the previous undo operation (Redo)
<CTRL> + S	Saves the current document
<CTRL> + O	Opens a saved document
<CTRL> + N	Creates a new editor document
<HOME>	Moves the caret to the beginning of the current line
<END>	Moves the caret to the end of the current line
<CTRL><HOME>	Moves to the beginning of the document
<CTRL><END>	Moves to the end of the document
<Backspace>	Deletes the previous character before the caret
<Delete>	Deletes the next character after the caret
<TAB>	Tab selected text over by one position

<SHIFT><TAB>	Removes tabs from the beginning of each selected line (de-tab)
F4	Executes the currently built project
F5	Compiles the source file currently in view
<CTRL> + F5	Rebuilds the currently opened project
F8	Builds a single file executable
<CTRL> + F8	Compiles the current project or source file and executes
<CTRL><SHIFT> + F8	Compiles the resource file of a project
<CTRL> + Keypad+	Magnify text size.
<CTRL> + Keypad-	Reduce text size.
<CTRL> + Keypad/	Restore text size to normal.
<CTRL><TAB>	Cycle through recent files.
<CTRL><Backspace>	Delete to start of word.
<CTRL><Delete>	Delete to end of word.
<CTRL><SHIFT><Backspace>	Delete to start of line.
<CTRL><SHIFT><Delete>	Delete to end of line.
<CTRL><HOME>	Go to start of document.
<CTRL><SHIFT><HOME>	Extend selection to start of document.
<ALT><HOME>	Go to start of display line.
<ALT><SHIFT><HOME>	Extend selection to start of display line.
<CTRL><END>	Go to end of document.
<CTRL><SHIFT><END>	Extend selection to end of document.
<ALT><END>	Go to end of display line.
<ALT><SHIFT><END>	Extend selection to end of display line.
<CTRL> + Keypad*	Expand or contract a fold point.
F2	Find previous selection
<SHIFT> + F2	Find first
F3	Find next selection.
<SHIFT> + F3	Find last.
<CTRL> + L	Line cut.
<CTRL><SHIFT> + T	Line copy.
<CTRL><SHIFT> + L	Line delete.
<CTRL> + T	Line transpose with previous.

<CTRL> + [Previous paragraph. Shift extends selection.
<CTRL> +]	Next paragraph. Shift extends selection.
<CTRL> + Left	Previous word. Shift extends selection.
<CTRL> + Right	Next word. Shift extends selection.
<CTRL> + /	Previous word part. Shift extends selection
<CTRL> + \	Next word part. Shift extends selection.

There are other shortcuts available for use. Explore the [Main Menu](#) option of the IDE to determine shortcuts to commonly used menu options.

Right-clicking in the *Work Area* of a *Code Editor* opens the popup menu shown below:

Cut	Ctrl-X
Copy	Ctrl-C
Paste	Ctrl-V
Select All	Ctrl-A
Find	Ctrl-F
Find First	Shft+F2
Find Prev	F2
Find Next	F3
Find Last	Shft+F3
Replace	Ctrl-H
Find in Help...	F1
Comment	F6
UnComment	Shft-F6
Insert File into Project	

Each available option is described in the table below.

Option	Description
Cut	Used to delete selected text from the currently selected source file in the WorkSpace . Disabled when there is no text selected. This option has a corresponding Main ToolBar button  . See Editing Text for additional information.
Copy	Used to copy selected text in the currently selected source file in the WorkSpace to the clipboard. Disabled when there is no text selected. This option has a corresponding Main ToolBar button  . See Editing Text for additional information

Paste	Used to paste/insert the contents of the clipboard in place of the currently selected text or at the current caret location in the currently selected source file in the WorkSpace . Disabled when clipboard is empty. This option has a corresponding Main ToolBar button  . See Editing Text for additional information.
Select All	Used to select all the text in the currently selected source file in the WorkSpace . See Editing Text for additional information.
Find	Used to find text in the currently selected source file in the WorkSpace . Opens a "Find" dialog. See Finding Text for additional information.
Find First	Used to find the first instance of the current search term in the currently selected source file in the WorkSpace . Disabled when there is no current search term in the currently selected source file. See Finding Text for additional information.
Find Prev	Used to find the previous instance of the current search term in the currently selected source file in the WorkSpace . Disabled when there is no current search term in the currently selected source file. See Finding Text for additional information.
Find Next	Used to find the next instance of the current search term in the currently selected source file in the WorkSpace . Disabled when there is no current search term in the currently selected source file. See Finding Text for additional information.
Find Last	Used to find the last instance of the current search term in the currently selected source file in the WorkSpace . Disabled when there is no current search term in the currently selected source file. See Finding Text for additional information.
Replace	Used to replace text in the currently selected source file in the WorkSpace . Opens a "Replace" dialog. See Replacing Text for additional information.
Find in Help...	Used to search the help files for the currently highlighted text in the currently selected source file. See the Utilities»Help Menu Editor for additional information.
Comment	Used to comment out one or more lines of code. See Commenting for details.
UnComment	Used to uncomment one or more previously commented lines of code. Disabled when the caret is not in a commented location. See Commenting for details.
Insert File in Project	Used to add the currently selected file to the current project. Disabled when there is no current project or if there is a current project but the file is already part of the project. See the How-To»Projects»Add Files section for additional information

Title Bar



A *Code Editor Title Bar* will contain the icon registered to the file's name extension on the User's computer. That is followed by the full name of the file minus any path information. If the *Code Editor* contains any unsaved edits that will be indicated by an asterisk following the file name. Each window also contains the standard system menu buttons to minimize, maximize, and close the

window.

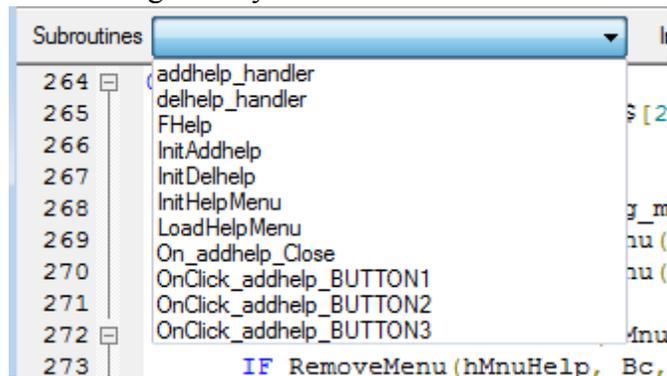
Toolbar



A *Code Editor Toolbar* contains three functional areas. The first two areas consists of drop-down combo boxes.

The first (shown below) contains all the names of subroutines in the file. Clicking an entry will cause the *Code Editor* window to scroll to the selected subroutine.

The second contains a list of all the files that are the object of \$INCLUDE directives in the current file. Clicking an entry will cause the selected file to be opened in a new *Code Editor* window.



Bookmarks

The third area of *Code Editor Toolbar* supports the use of bookmarks. Bookmarks are a means by which the User can mark a line in the code and easily return there at some later time. Refer to the image below for the following discussion.

To place a bookmark the User clicks anywhere on the desired line and then clicks the  button on the *Code Editor Toolbar*. This will result in two things:

1. A red circle will appear in the margin of the line that has been bookmarked.
2. The line number will be added to the drop-down list labeled "Jump to Line". This will occur regardless of whether line numbers are being displayed or not.

To return to a bookmarked line, the User selects the desired line number from the list and then clicks the  button. The window will scroll until the desired line is in view.

When a bookmark is no longer required the User can select it from the list and then click the  button. To remove all the bookmarks at one time click the  button.

If the User closes the file and then reopens it at some later date all bookmarks will be restored to the state they were in the last time the file was closed. The information is stored in a file with the same base name as the source file plus the ".ixb" extension.

```

82  ISTRING temp$[260]="
83  STRING cfname = LTRIM$(RTRIM$(GETCONTROLTEXT (addhelp,addhelp_ED
84  STRING helpname= LTRIM$(RTRIM$(GETCONTROLTEXT (addhelp,addhelp_EDIT2)))
85  ISTRING helpexename[260]= LTRIM$(RTRIM$(GETCONTROLTEXT (addhelp,addhelp_EDIT3)))
86  IF cfname=""
87  MESSAGEBOX addhelp,"A config file name for the Help file is required.\nPlease enter a w
88  RETURN FALSE
89  ENDIF
90  ISTRING bpath[260]=GETSTARTPATH
91  'GET ADDON HELP FILES
92  SplitTwo(bpath,"\\",bpath,temp$,1,1)
93  SplitTwo(bpath,"\\",bpath,temp$,1,1)
94  bpath+="\\help2\\"
95  IF DirExists(bpath) =1
96
97  ELSE
98  bpath=GETFOLDERPATH(@CSIDL_LOCAL_APPDATA)+"\\IonicWind\\IWBasic3\\help2\\"
99  ENDIF

```

Code Folding

The *Code Editor* automatically assigns fold points when code is entered based upon language keywords. These fold points are identified in the margin with small squares with either a plus or minus sign in the margin. When a code block is expanded (all lines of code shown) a minus sign is in the box and there is a vertical line connected to the bottom of the box indicating the span of the code block. (See example in previous image.) If the User clicks the box the block of code is collapsed to only show the first line of code in the block. The minus sign in the small box will change to a plus sign. Subsequent clicking of the small box will expand the code back out.

The right-most button on the *Code Editor Toolbar*, , is used to toggle the folding of all fold points in the source file at one time. The image below shows a typical source file with all code blocks collapsed.

Note: **\$REGION** and **\$ENDREGION** can be used by the User to define custom code blocks that can be collapsed.

```

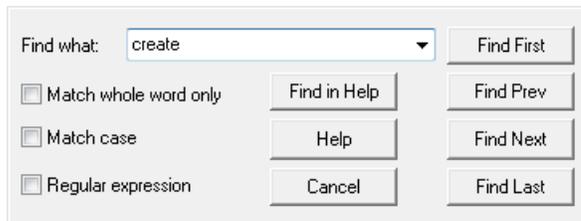
18  CONST addhelp_BUTTON2 = 0X108
19  CONST addhelp_BUTTON3 = 0X109
20
21  GLOBAL SUB InitAddhelp()
36  SUB addhelp_handler(),INT
67  SUB On addhelp Close(),INT
72  SUB OnClick addhelp_BUTTON1(),INT
81  SUB OnClick addhelp_BUTTON2(),INT
125  SUB OnClick addhelp_BUTTON3(),INT
130  CONST STATIC_1 = 1
131  CONST LISTBOX_2 = 2
132  CONST BUTTON_3 = 3
133  CONST BUTTON_4 = 4
134  GLOBAL SUB InitDelhelp()
143  SUB delhelp_handler(),INT
194  GLOBAL SUB InitHelpMenu()
221  SUB FHelp(STRING bpathx)
264  GLOBAL SUB LoadHelpMenu()

```

Finding Text

Anytime the User performs any of three actions, when there is a *Code Editor* window open, the *Find* dialog (shown below) will open. Those three actions are:

- Pressing <CTRL>+F on the keyboard.
- Selecting the [Main Menu](#) *Editor/Find* option.
- Right-clicking in the *Code Editor* and selecting *Find* from the popup menu.



Each item in the dialog is described below.

Option	Description
Find What	The term to search for. See the discussion following this table.
Match whole word only	Find stand-alone words or also terms embedded in larger text strings.
Match case	Find search term as typed or ignore case.
Regular expression	When checked it indicates that search term contains wildcard characters.
Find First	Used to find the first instance of the current search term in the currently selected source file in the WorkSpace . Disabled when there is no current search term. If found the <i>Code Editor</i> window is scrolled so the found text is visible and highlighted. <SHFT>F2 performs the same action when a term is highlighted in the <i>Code Editor</i> window.
Find Prev	Used to find the previous instance of the current search term in the currently selected source file in the WorkSpace . Disabled when there is no current search term. If found the <i>Code Editor</i> window is scrolled so the found text is visible and highlighted. F2 performs the same action when a term is highlighted in the <i>Code Editor</i> window.
Find Next	Used to find the next instance of the current search term in the currently selected source file in the WorkSpace . Disabled when there is no current search term. If found the <i>Code Editor</i> window is scrolled so the found text is visible and highlighted. F3 performs the same action when a term is highlighted in the <i>Code Editor</i> window.

Find Last	Used to find the last instance of the current search term in the currently selected source file in the WorkSpace . Disabled when there is no current search term. If found the <i>Code Editor</i> window is scrolled so the found text is visible and highlighted. <SHFT>F3 performs the same action when a term is highlighted in the Code Editor window.
Find in Help...	Used to search the help files for the search term. See the Utilities»Help Menu Editor section for additional information. Disabled when there is no current search term. F1 performs the same action when a term is highlighted in the Code Editor window.
Help	Opens the <i>Regular Expression Help</i> window, shown below.
Cancel	Closes the <i>Find</i> dialog.

The *search term* can be established in multiple ways.

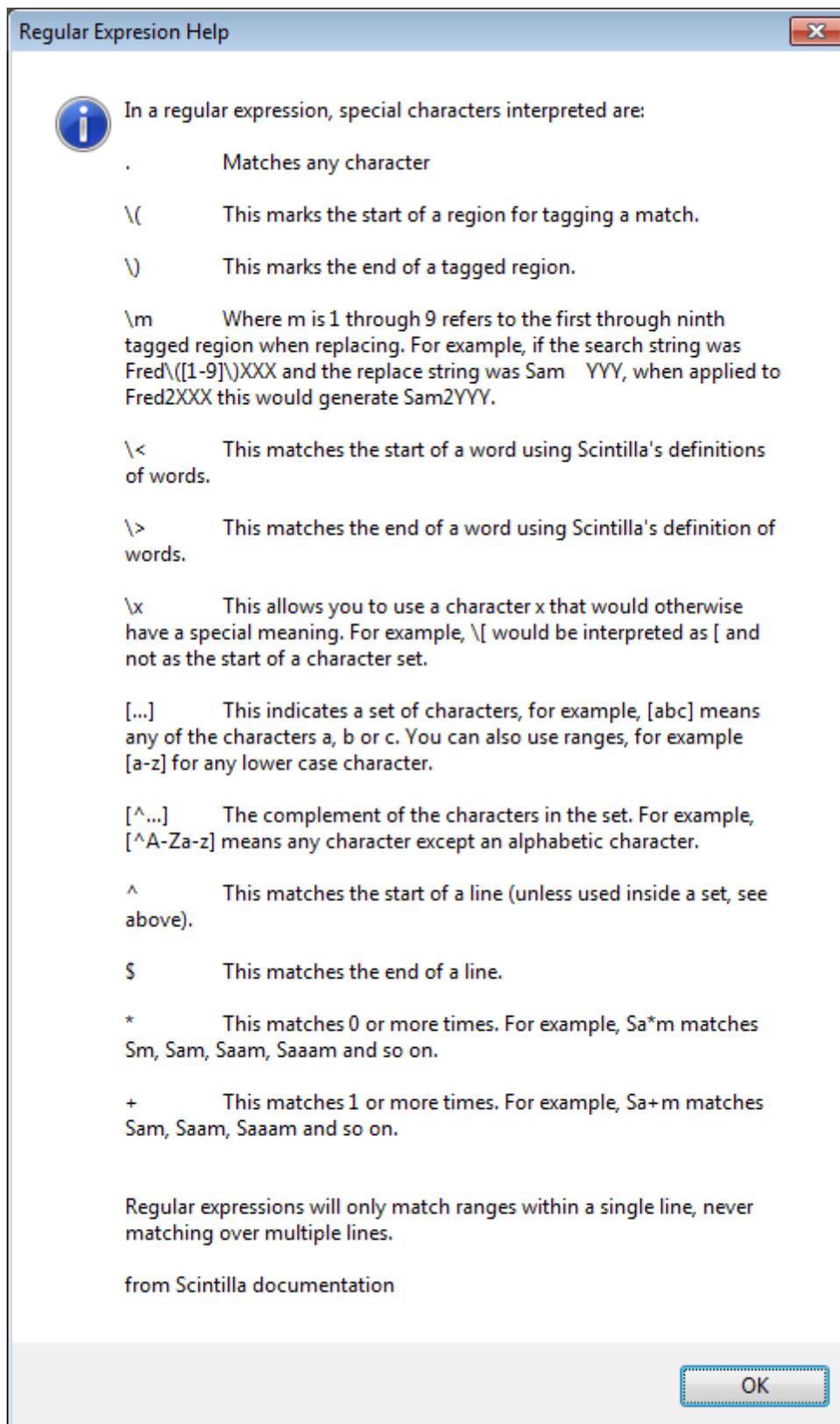
If a text string is selected in the *Code Editor* when the *Find* dialog is opened then the selected text automatically becomes the *search term*.

If there is no selected text when the *Find* dialog is opened the *search term* will be the last term searched for if one exists otherwise the *search term* will be blank.

At any time the User may enter a new *search term* in the *Find* dialog.

The *search term* is stored in a global variable. This allows the User to search for a term in one *Code Editor* and then select a different *Code Editor* and search for the same term without having to re-enter it.

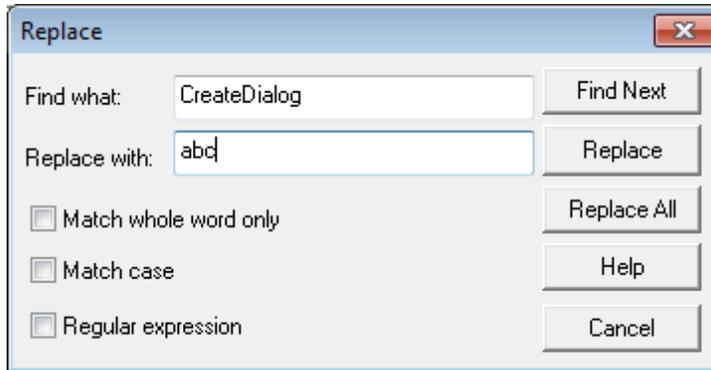
Regular Expression Help



Replacing Text

Anytime the User performs any of three actions, when there is a *Code Editor* window open, the *Replace* dialog (shown below) will open. Those three actions are:

- Pressing <CTRL>+H on the keyboard.
- Selecting the [Main Menu](#) *Editor/Replace* option.
- Right-clicking in the *Code Editor* and selecting *Replace* from the popup menu.



Each item in the dialog is described below.

Option	Description
Find What	The term to search for. See the discussion of <i>search term</i> following the previous table above which also applies to the <i>Replace search term</i> ..
Replace with	Text to use when replacing. The <i>replacement text</i> .
Match whole word only	Find stand-alone words or also terms embedded in larger text strings.
Match case	Find search term as typed or ignore case.
Regular expression	When checked it indicates that search term contains wildcard characters.
Find Next	Used to find the next instance of the current search term in the currently selected source file in the WorkSpace . Disabled when there is no current search term. If found the <i>Code Editor</i> window is scrolled so the <i>found text</i> is visible and highlighted.
Replace	Used to replace the <i>found text</i> in the currently selected source file in the WorkSpace with the replacement text. Disabled when there is no <i>found text</i> .
Replace All	Used to automatically find all <i>found text</i> in the currently selected source file in the WorkSpace and replace each instance with the <i>replacement text</i> . Disabled when there is no current search term. The <i>Code Editor</i> window is scrolled so the last replacement is visible and highlighted.

Help	Opens the <i>Regular Expression Help</i> window, shown above.
Cancel	Closes the <i>Replace</i> dialog.

Commenting

The Commenting features of the Code Editor make easy to comment and/or un-comment portions of the User's code. As far as the Code Editor is concerned comments are either:

- Full single line comments, or
- Partial line or multi-line comments

Full single line comments are marked with a "' ' " at the beginning of the line.

Using the following screenshot assume that the User right-clicks anywhere on line # 212.

The popup menu opens and the *Comment* option is enabled since the line is not currently commented.

```

201     ISTRING filename[260]=" "
202     'GET DISTRIBUTION HELP FILES
203     bpath=GETSTARTPATH
204     FHelp(bpath)
205     'GET ADDON HELP FILES
206     SplitTwo(bpath,"\\",bpath,temp$,1,1)
207     SplitTwo(bpath,"\\",bpath,temp$,1,1)
208     bpath+="\\help2\\"
209     IF DirExists(bpath) =1
210     |     FHelp(bpath)
211     |
212     |     ELSE
213     |         bpath=GETFOLDERPATH(@CSIDL_LOCAL_APPDATA )+"\\IonicWind\\IWBasic3\\help2\\"
214     |         IF DirExists(bpath)=1
215     |         |     FHelp(bpath)
216     |         |
217     |         |     ENDIF
218     |         ENDIF
219     |         LoadHelpMenu()
220     |     ENDIF
221     |
222     |     RETURN

```

Clicking on the *Comment* option will result in line # 212 being commented as indicated.

```

201     ISTRING filename[260]=" "
202     'GET DISTRIBUTION HELP FILES
203     bpath=GETSTARTPATH
204     FHelp(bpath)
205     'GET ADDON HELP FILES
206     SplitTwo(bpath,"\\",bpath,temp$,1,1)
207     SplitTwo(bpath,"\\",bpath,temp$,1,1)
208     bpath+="\\help2\\"
209     IF DirExists(bpath) =1
210     |     FHelp(bpath)
211     |
212     |     '     BPATH=GETFOLDERPATH(@CSIDL_LOCAL_APPDATA )+"\\IONICWIND\\IWBASIC3\\HELP2\\"
213     |     IF DirExists(bpath)=1
214     |     |     FHelp(bpath)
215     |     |
216     |     |     ENDIF
217     |     |
218     |     |     ENDIF
219     |     |     LoadHelpMenu()
220     |     |
221     |     |     RETURN

```

To remove this type of comment simply right-click anywhere on line # 212 and select the *UnComment* option which is now enabled.

The line will return to its original uncommented state.

If there is any selected text when the User right-clicks in any portion of the selected text, the popup menu opens and the *Comment* option is enabled since the line is not currently commented. The screenshot below indicates that lines # 209 thru # 216 have been selected.

```

201 | ISTRING filename[260]=" "
202 | 'GET DISTRIBUTION HELP FILES
203 | bpath=GETSTARTPATH
204 | FHelp (bpath)
205 | 'GET ADDON HELP FILES
206 | SplitTwo (bpath, "\\ ", bpath, temp$, 1, 1)
207 | SplitTwo (bpath, "\\ ", bpath, temp$, 1, 1)
208 | bpath+="\\help2\\"
209 | IF DirExists (bpath) =1
210 |     FHelp (bpath)
211 | ELSE
212 |     bpath=GETFOLDERPATH (@CSIDL_LOCAL_APPDATA )+"\\IonicWind\\IWBasic3\\help2\\"
213 |     IF DirExists (bpath)=1
214 |         FHelp (bpath)
215 |     ENDIF
216 | ENDIF
217 | LoadHelpMenu ()
218 | RETURN

```

Clicking on the *Comment* option will result in all the selected text being commented as indicated below.

```

201 | ISTRING filename[260]=" "
202 | 'GET DISTRIBUTION HELP FILES
203 | bpath=GETSTARTPATH
204 | FHelp (bpath)
205 | 'GET ADDON HELP FILES
206 | SplitTwo (bpath, "\\ ", bpath, temp$, 1, 1)
207 | SplitTwo (bpath, "\\ ", bpath, temp$, 1, 1)
208 | bpath+="\\help2\\"
209 | /* IF DIREXISTS (BPATH) =1
210 |     FHELP (BPATH)
211 | ELSE
212 |     BPATH=GETFOLDERPATH (@CSIDL_LOCAL_APPDATA )+"\\IONICWIND\\IWBASICS3\\HELP2\\"
213 |     IF DIREXISTS (BPATH)=1
214 |         FHELP (BPATH)
215 |     ENDIF
216 | ENDIF*/
217 | LoadHelpMenu ()
218 | RETURN

```

To remove this type of comment simply right-click anywhere within the commented text and select the *UnComment* option which is now enabled.

The commented text will return to its original uncommented state.

NOTE: The *Comment* and *UnComment* options are mutually exclusive. As a result the nesting of comments is not allowed.

However there is a situation where a nested relationship can occur. If the User selects a block of lines to comment and one of those lines is already a single line comment the *Comment* option will be active provided the User does not right-click on the commented line.

After that situation is established, if the User then right-clicks on the single line comment and selects *UnComment* only that line will be uncommented.

But it will remain commented because it is still inside the block comment.

Had the User right clicked on any other line in the commented block and chose *UnComment* the entire block would be uncommented but the single line comment would remain in place.

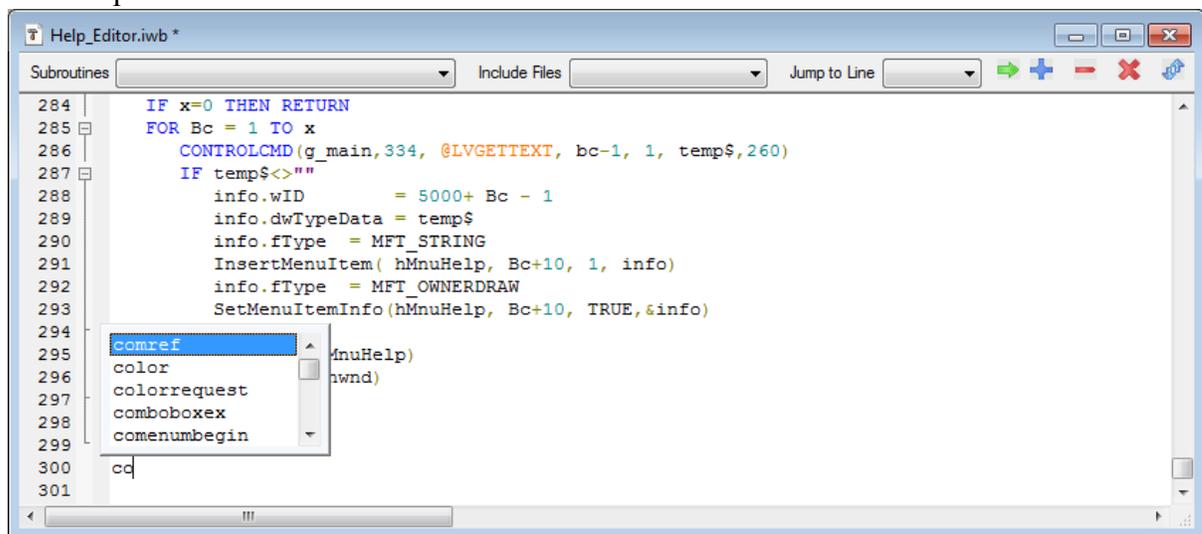
AutoComplete

The *AutoComplete* feature is an option the User can turn on/off. See the [How-To»Set Editor Preferences](#) section for details.

When the option is ON the User's keyboard entries are monitored. When the User types the first two or more letters of a word the *Code Editor* will check to see if there are any language keywords that start with the same letters. If there are then a listview will open showing the keywords that are possible matches. The list will change with each added letter, refining the list. If the User is indeed intending to type a keyword simply clicking on the desired word in the list will cause the rest of the word to be automatically typed and the list closed.

If the word is not in the list the list will close when no possible matches exists or the User types a non letter character.

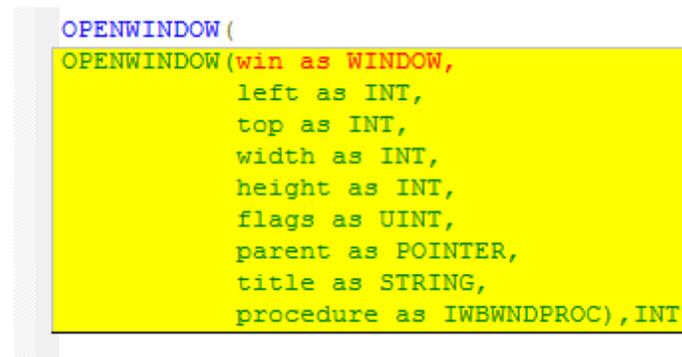
An example is shown below.



AutoTip

The *AutoTip* feature is an option the User can turn on/off. See the [How-To»Set Editor Preferences](#) section for details.

When the option is ON the User's keyboard entries are monitored. When the User types a language defined function or command name followed by a "(" an *AutoTip* window will appear as shown below:



The entire function syntax is shown with the current parameter to be entered highlighted. When the User enters the parameter (variable or constant) and the following " , " the *AutoTip* will shift the highlight to the next required parameter.

The image below shows that five parameters have been entered and is awaiting the sixth parameter, "flags".

```
OPENWINDOW(w1,0,0,500,300,  
OPENWINDOW(win as WINDOW,  
    left as INT,  
    top as INT,  
    width as INT,  
    height as INT,  
    flags as UINT,  
    parent as POINTER,  
    title as STRING,  
    procedure as IWBWNDPROC),INT
```

When the last parameter is entered the User enters a ")" and the *AutoTip* window closes.

NOTES:

- In order for *AutoTip* feature to work the "(" and ")" must be used even when it is not required.
- Once a parameter and its ", " has been entered there can be no "backing up" and retyping.
- If the User does anything to cause the *Code Editor* to lose focus the *AutoTip* will close. Once closed the only way to get the *AutoTip* to reappear is to delete all the parameters, including the parenthesis and retype starting with "(".

The *AutoTip* window background color, normal text color and highlight color are all selectable. See the [How-To»Set Editor Preferences](#) section for details.

Editing Text

To select text for copying and cutting place the mouse cursor (I-Beam) over the first character of the text to select. Press the left mouse button down and 'drag' the cursor over the text to select. If you drag past the confines of the window it will automatically scroll allowing the continued selection of text until the beginning or the end of the document.

A column of text can be selected by doing the above while holding down the <ALT> key.

If any key is pressed when text is selected then the selected portion will be replaced with the new text. The same will occur if the User right-clicks and selects *Paste*, provided there is text saved to the clipboard. The selected portion can also be deleted by pressing the <Delete> key.

Text may also be selected using the keyboard by holding the SHIFT key down and pressing one of <HOME>, <END>, <PAGEUP>, <PAGEDOWN>, or the arrow keys.

Text may also be selected from the current caret position to the position clicked with the left mouse button by holding down the SHIFT key.

Also, clicking a word with the CONTROL key pressed will select the whole word.

Saving the file

The default extension for IW BASIC source files is .iwb and should not be changed to anything else. For include files the default extension is .INC.

See the [How-To»Files»Save a File](#) section for details on the various ways files can be saved.

Recognized file encoding

The editor recognizes 3 different encoding schemes:

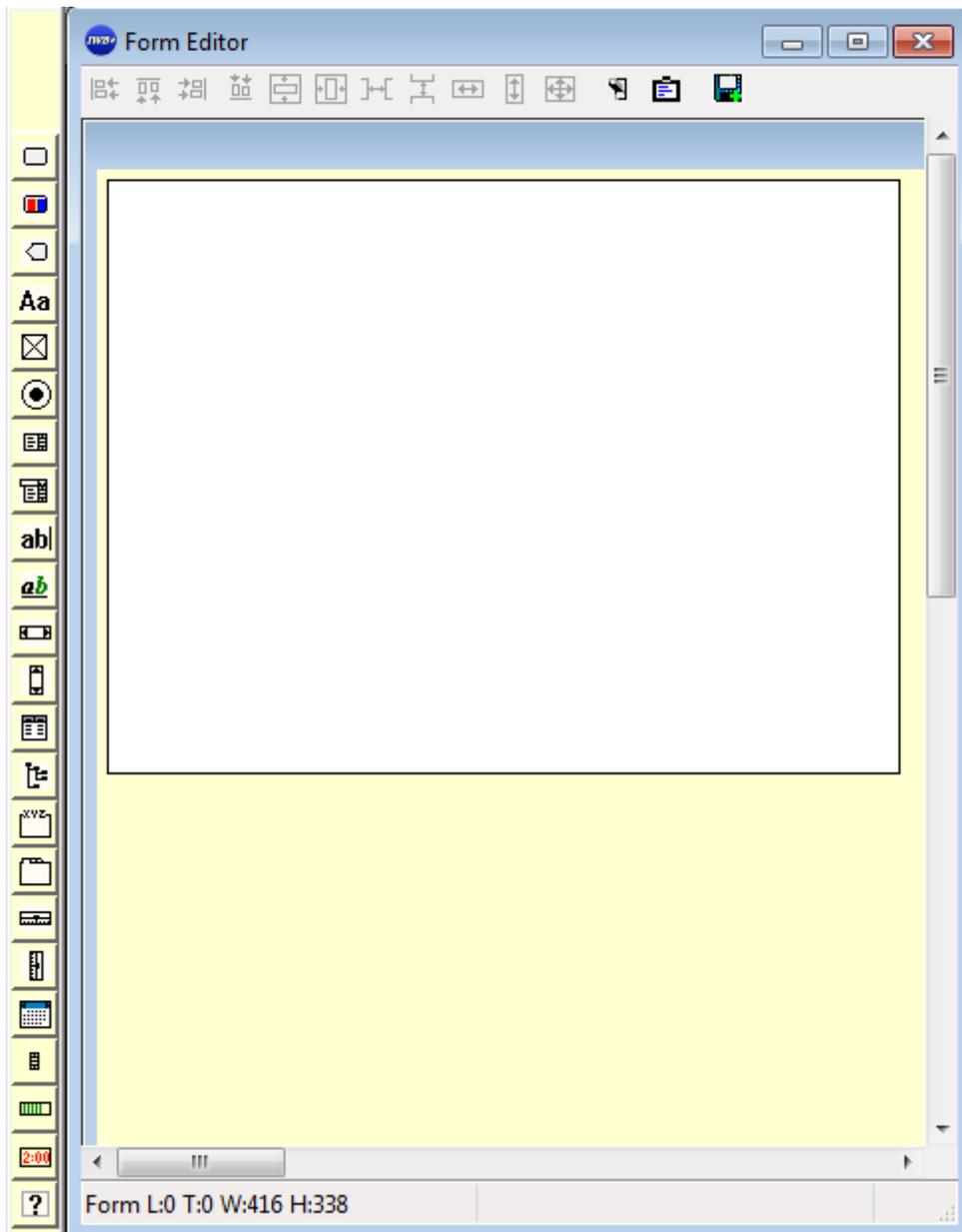
1. ASCII
2. UTF-8 - encoded strings are supported up to 6 bytes per character.
3. DBCS

6.2 Form Editor

Introduction

The *Form Editor* is used to layout controls, text and set the attributes of a dialog or window. Once a satisfactory layout is made the User can generate the source statements necessary to create the form. Optionally, the User can generate a complete skeleton program for the form. The User can create a new form or edit an existing form.

The *Form Editor* User interface is shown below. It consist of a [Main Toolbar](#), [Control Toolbar](#), [Status bar](#), and [Work area](#).



Main Toolbar

The Main Toolbar (shown below) contains 11 buttons for positioning/sizing controls and 2 buttons for testing/code generation.



The function of each button is described in the following table:

Button	Description
	Aligns the left edges of two or more selected controls. The first control selected is used as the reference. Disabled when less than two controls are selected.
	Aligns the top edges of two or more selected controls. The first control selected is

	used as the reference. Disabled when less than two controls are selected.
	Aligns the right edges of two or more selected controls. The first control selected is used as the reference. Disabled when less than two controls are selected.
	Aligns the bottom edges of two or more selected controls. The first control selected is used as the reference. Disabled when less than two controls are selected.
	Centers all selected controls vertically on the form. Disabled when no control is selected.
	Centers all selected controls horizontally on the form. Disabled when no control is selected.
	Equalizes the horizontal spacing between three or more selected controls. The left most and right most controls are used as the anchors. Disabled when less than three controls are selected.
	Equalizes the vertical spacing between three or more selected controls. The top most and bottom most controls are used as the anchors. Disabled when less than three controls are selected.
	Sets all selected controls to the same width. The first control selected is used as the reference. Disabled when less than two controls are selected.
	Sets all selected controls to the same height. The first control selected is used as the reference. Disabled when less than two controls are selected.
	Sets all selected controls to the same width and height. The first control selected is used as the reference. Disabled when less than two controls are selected.
	Displays a preview of the current form as it would appear in the User's application.
	Opens the <i>Source Generation Options</i> dialog (shown below).
	Opens the <i>Save As</i> dialog. The current form will be saved to the new file name and the <i>Form Editor</i> is closed. The User can abort the save and continue to edit the current form.

Control Toolbar

The controls in the *Control Toolbar* can be dragged and dropped on the current form. The following table list all the currently available controls.

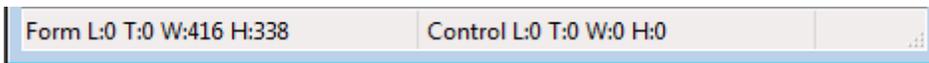
NOTE: All the controls are covered in detail in the [Windows Programming»Controls](#) section.

Control	Description
	The @SYSBUTTON control is a Windows XP Theme compatible button control. When using @SYSBUTTON you will not be able to set the color of the control manually with SETCONTROLCOLOR.
	The @BUTTON control can have the foreground and background colors changed. With

	the proper style flag the background can be a bitmap.
	The @RGNBUTTON control can have irregular shapes and multi-color backgrounds
Aa	The @STATIC control is used for static text.
	The @CHECKBOX control is used for selectable options.
	The @RADIOBUTTON control is used in groups of two or more for making exclusive choices.
	The @LISTBOX control is used for displaying a list of items. The User can select an item from the list
	The @COMBOBOX control is a combination of a Listbox and an Edit control.
abl	The @EDIT control is used for entering alphanumeric data.
ab	The @RICHEDIT control is an elaborate version of an Edit control.
	The @SCROLLBAR control with the @CTSCROLLHORIZ set is used to create a Horizontal Scrollbar.
	The @SCROLLBAR control with the @CTSCROLLVERT set is used to create a Vertical Scrollbar.
	The @LISTVIEW control is used to display a list of items in various formats. Each item can have additional information shown.
	The @TREEVIEW control is used to display items in a collapsible / expandable format with nesting.
	The @GROUPBOX control is used in dialogs to draw a labeled rectangle around a group of controls.
	Used to create a TabControl.
	Used to create a Horizontal Trackbar.
	Used to create a Vertical Trackbar.
	Used to create a Calendar control.
	Used to create a Spinner control.
	Used to create a Progressbar
	Used to create a Date/Time control.
	Used to create a placeholder for the User's custom control.

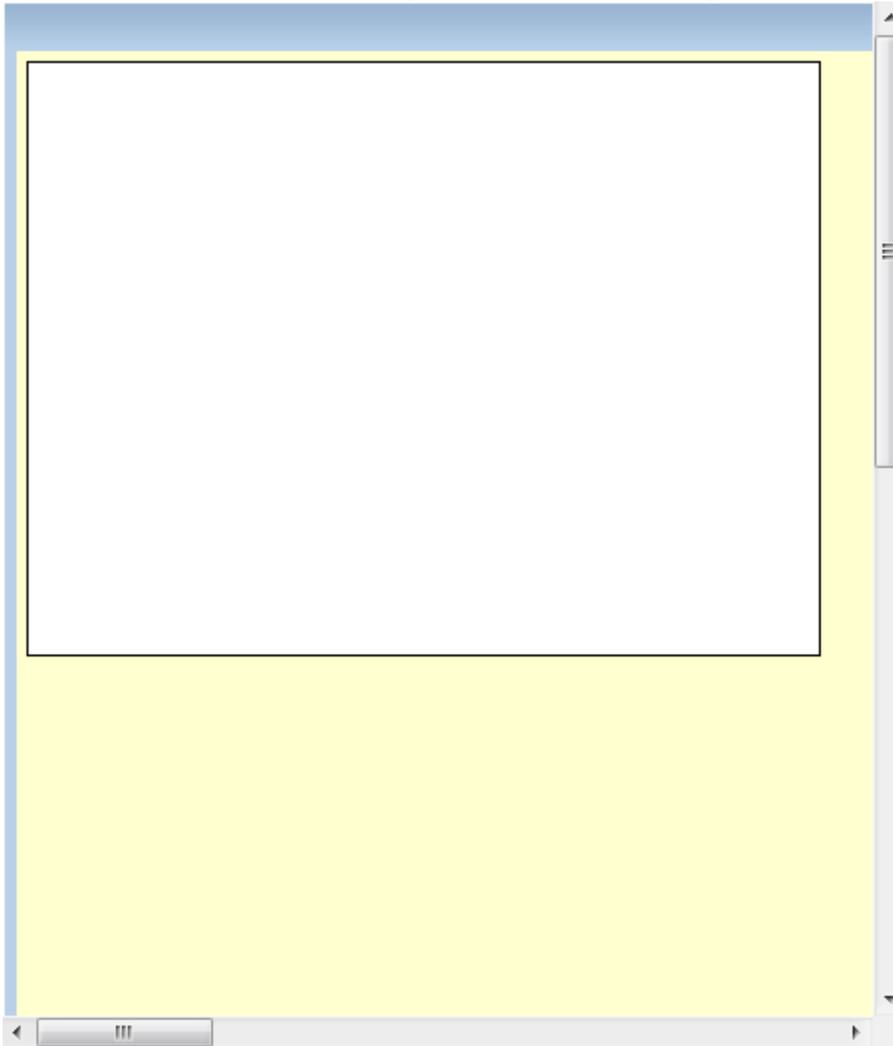
Status Bar

The *Status Bar* shows the overall size of the form which contains the client area currently displayed in the *Form Editor Work Area* plus space for any menu, statusbar and borders. It also shows the size and location of the currently selected control. When no control is selected the control text will disappear.



Work Area

The Work Area contains a rectangle (shown below) that represents the client area of the form.



Creating a new Form

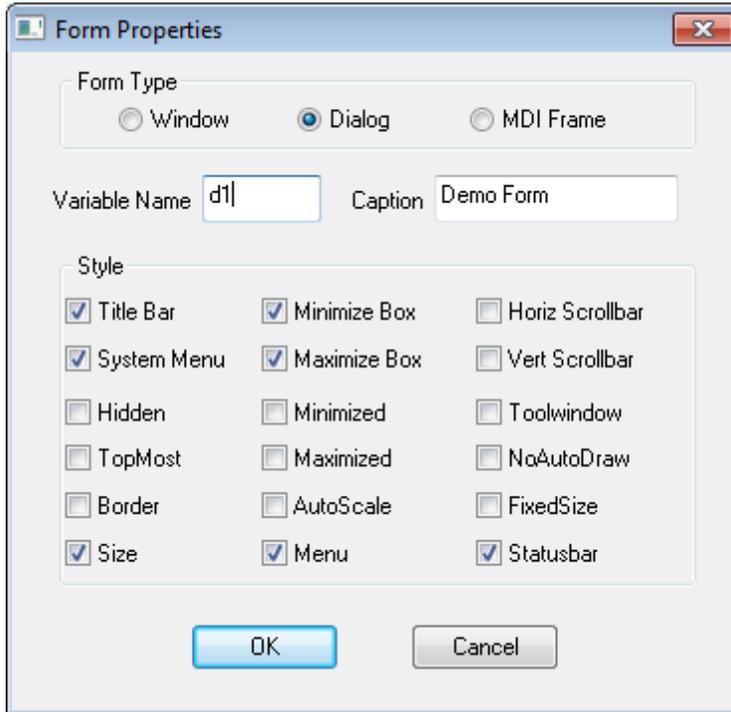
Select *Tools/Form Editor/New Form* from the [Main Menu](#). The *Form Editor* will open with a representation of a blank form, as shown above. The form can be saved by responding with *Yes* to the *Save Form* dialog when the Form Editor is closed..

Sizing the Form

Click on any blank area in the form rectangle to show the sizing handles. Drag the sizing handles to change the form to the desired size.

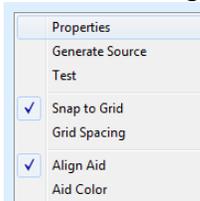
Form Properties

Double click any blank area in the form to bring up the *Form Properties* dialog (shown below).



Here you can change the form type, caption, variable used and styles of the form. The *variable* entry should correspond to the one used in your program defined with DEF / DIM ... as DIALOG or WINDOW. This variable name will be used as a prefix for the id names assigned to controls as they are added to the form.

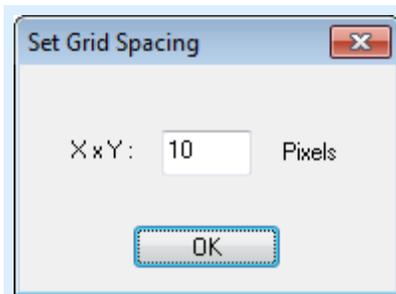
You can also right click on any blank area in the form to bring up the following popup menu:



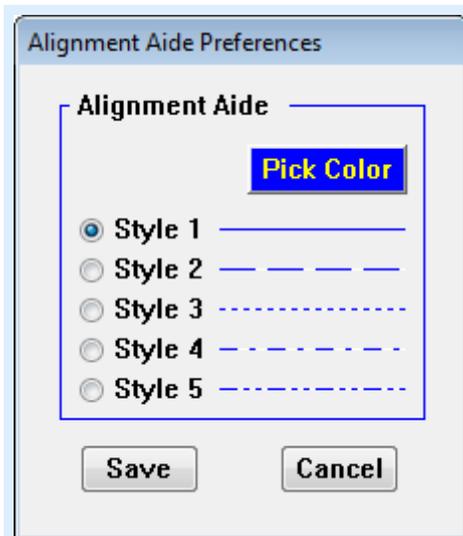
The 7 available options are described in the table below.

Option	Description
Properties	Opens the <i>Form Properties</i> dialog (shown above). Same as double clicking any blank area in the form.
Generate Source	Opens the <i>Source Generation Options</i> dialog (shown below). Same as clicking  in the main toolbar.
Test	Displays a preview of the current form as it would appear in the User's application. Same as clicking  in the main toolbar.
Snap to Grid	The Form Editor has a hidden grid that may be used to facilitate the layout of controls on a form.

	When checked, the selection of a control (or any movement of a control) will result in its upper left corner being aligned with the nearest grid point. The alignment point will be the nearest grid point above and left of the current location of the control's upper left corner. If the current upper left corner is already on a grid point no adjustment will be made.
Grid Spacing	Opens the <i>Set Grid Spacing</i> dialog, shown below. Disabled when <i>Snap to Grid</i> is not checked.
Align Aid	When this option is activated every unselected control will have its outline drawn in the desired color and style. This facilitates the alignment of controls relative to each other when it normally has no visible border (@static, @radiobutton) or the edges of the control are hard to discern.
Aid Color	Opens the <i>Alignment Aide Preferences</i> dialog, shown below. Disabled when <i>Align Aid</i> is not checked.



Allows the User to set the default spacing of the grid points in pixels.



When the *Align Aid* option is activated this dialog is used to set the color and style of the outline drawn around every unselected control.
The functions are described in the following table.

Button	Description
--------	-------------

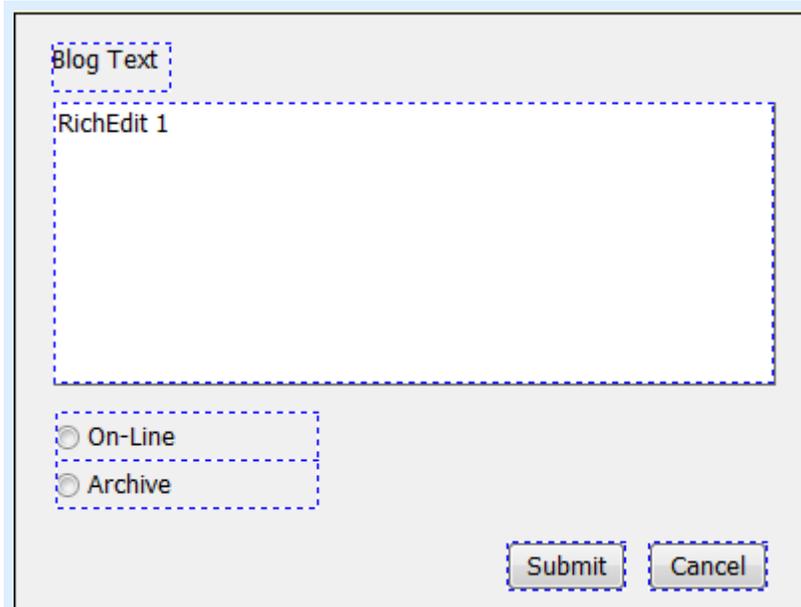
Pick Color	When the button is clicked a standard <i>Color Picker</i> dialog is opened allowing the User to select the desired color for the outline box. The background color of the button reflects the currently selected color.
Style 1 thru Style 5	Allows the User to select the desired line style for the outline box. To the right of the radiobuttons are examples of the corresponding line style drawn in the currently selected color. See examples below.
Save	Saves the current selections, closes the dialog, and immediately applies the selections to the current form.
Cancel	Closes the dialog, discarding any changes that may have been made.

The screenshot shows a dialog box titled "Blog Text". It contains a large text area labeled "RichEdit 1". Below the text area are two radio buttons: "On-Line" and "Archive". At the bottom right are two buttons: "Submit" and "Cancel".

Shows a typical form with *Align Aid* unchecked.

The screenshot shows the same dialog box as above, but with a blue border around the "RichEdit 1" text area and the radio buttons. The "On-Line" radio button is selected.

Shows a typical form with *Align Aid* checked and *Style 1* selected.

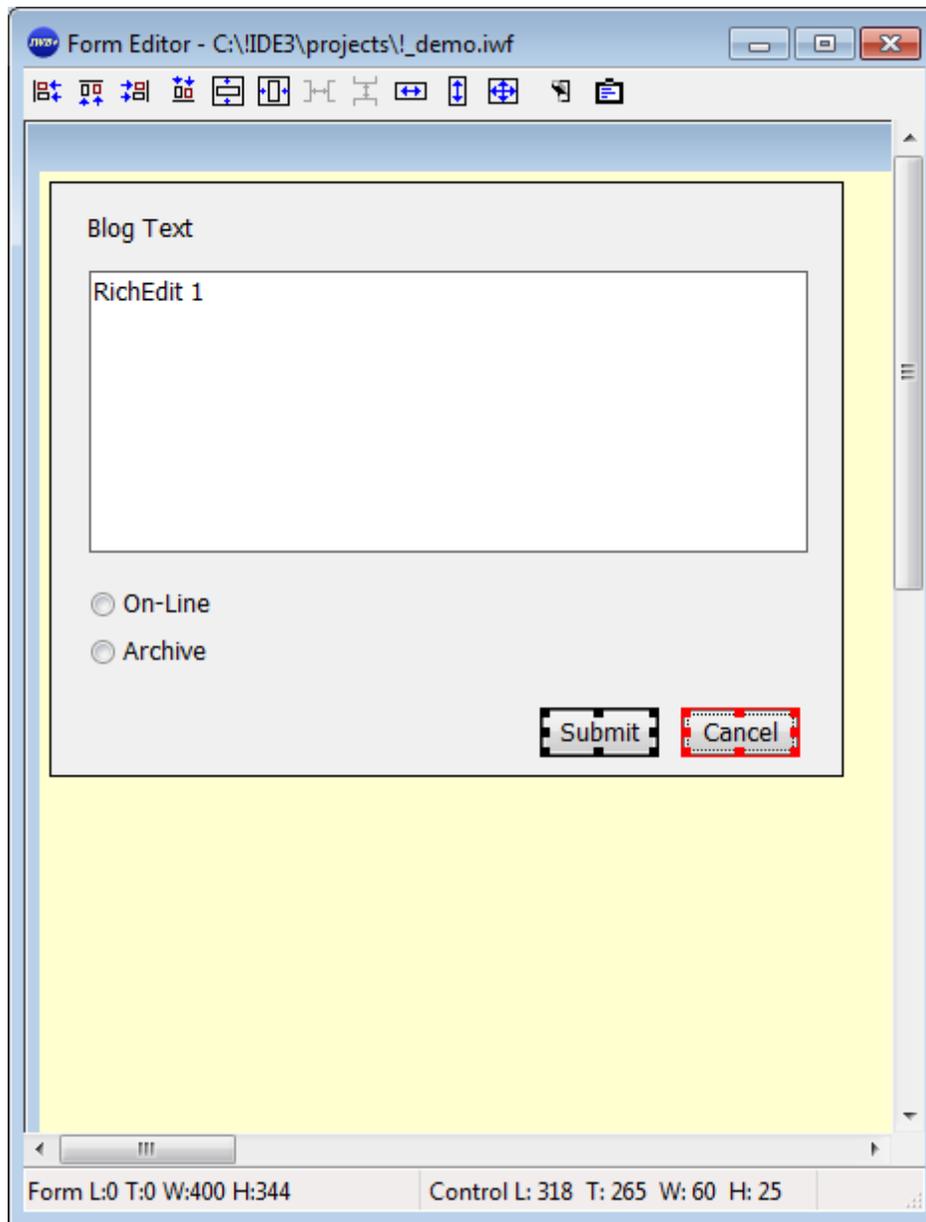


The screenshot shows a form with a light gray background. At the top left, there is a text box labeled "Blog Text". Below it is a large text area labeled "RichEdit 1". At the bottom left, there are two radio buttons: "On-Line" and "Archive". At the bottom right, there are two buttons: "Submit" and "Cancel". All controls are enclosed in dashed blue boxes, indicating they are selected or highlighted.

Shows a typical form with *Align Aid* checked and *Style 3* selected.

Adding Controls

Click any control button on the [Control Toolbar](#) and drag it to the desired location on the form. Most controls will display their type and sequence number as the caption to aid in identifying controls on a form. That is why the RichEdit control below contains 'RichEdit 1'. The other controls have had their captions modified in the example.



Moving and Sizing controls

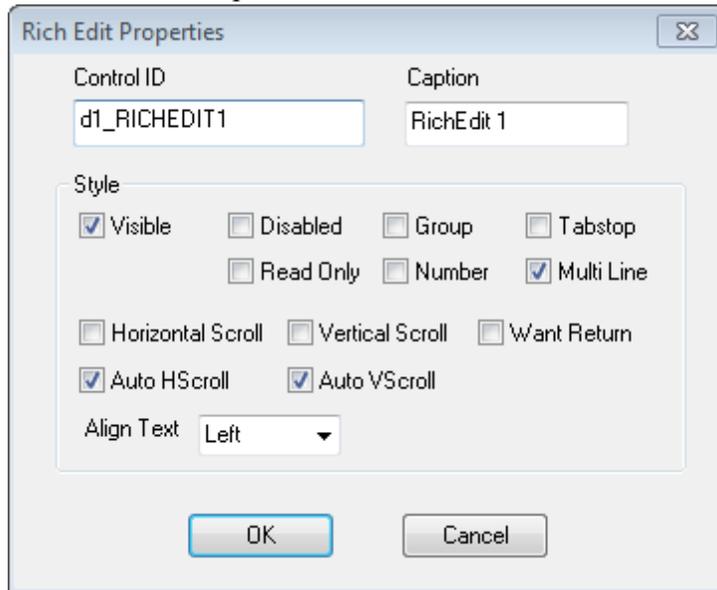
Clicking on any control will show the sizing handles. You can drag the control anywhere in the dialog. The [Status Bar](#) will update in realtime to indicate the size and location of the control. The selected control can also be moved using the keyboard arrow keys. The [Main Toolbar](#) contains additional control manipulation functions.

Select multiple controls by holding down the control <CTRL> key and selecting other controls with the mouse. The first control selected will be shown in a black border and all others in red as shown in the example above. The [Main Toolbar](#) contains additional control manipulation functions that use the first selected control as a reference. *Align Left* for example will align the left border of all the selected controls with the first selected. The *Evenly Space* actions require 3 or more selected controls. The [Status Bar](#) will update in realtime to indicate the size and location of the last clicked on

control. The selected controls can also be moved, as a group, using the keyboard arrow keys.

Changing control properties

Double click on any control, except the Calendar control, to bring up the *Property* dialog. Each control type has its own *Property* dialog. Shown below is the *Property* dialog for the RichEdit control in the example.



The control is automatically assigned an ID when it is initially created. It consists of the form name variable name when the control was created + the control type + a sequence number. The User may change that ID to something else if desired. The only requirement is that the ID has to follow the IWBASIC rules for constants.

You can also right click on a control to bring up a menu to access the control *Properties* or *Delete* the control.

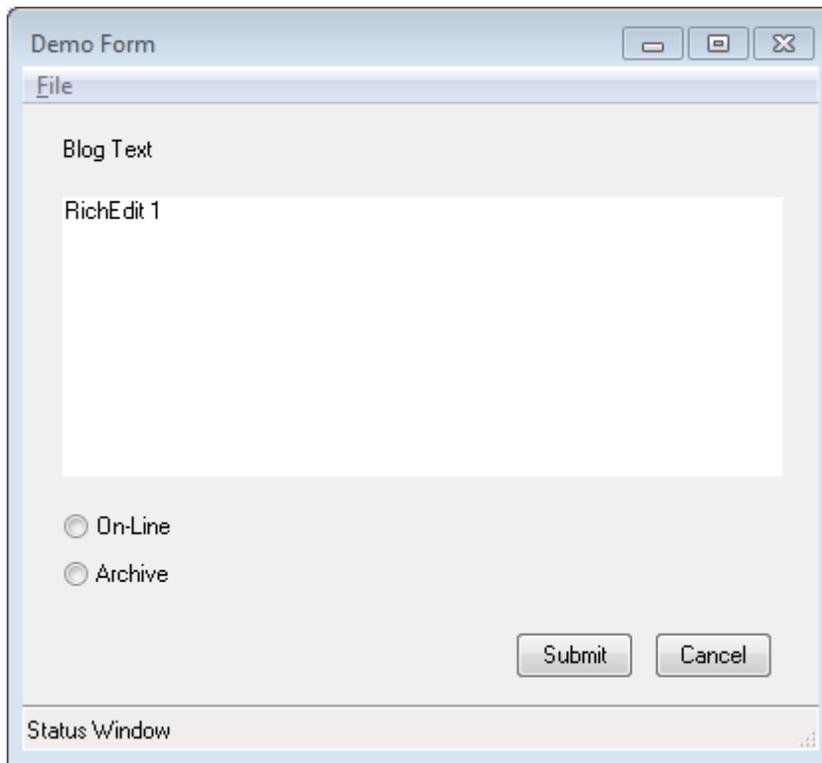
Details about the options available for each control are covered in the [Windows Programming»Controls](#) section.

Note:

Traditionally an ID of 1 is reserved for OK buttons and an ID of 2 is reserved for Cancel buttons. losing a dialog by pressing the <ESC> key will send a control ID of 2 indicating the user wishes to cancel the dialog.

Testing the Form

To test the Form right-click in a blank area of the form and choose *Test* or click on the  button of the [Main Toolbar](#). If you create a form with no close box you can close the test dialog by pressing the escape <ESC> key. The displayed test of the above example is shown below.



Notice that the test preview of our form has a menu bar and a status bar and they did not appear in the [Work Area](#) while the form was being designed. These two options were selected in the Properties dialog for the form.

The Form Editor made the necessary adjustments to the overall form size to account for the options while maintaining the size of the client area of the form.

Generating source code

Once the User has completed the design of the form and is satisfied with its test appearance it is time to generate the source code.

The User initiates the process by either right-clicking in a blank area of the form and selecting *Generate Code* or by clicking the  button on the [Main Toolbar](#). Each action will result in the Source Generation Options dialog opening (shown below).



The User has the option of copying the source code to the system clipboard, or opening in a new [Code Editor](#) window in the [Workspace](#).

The following is the basic code that is generated for our example from above.

```

1  DIALOG d1
2  |
3  ▣ ENUM d1_const
4      win_BUTTON1 = 100
5      win_BUTTON2
6      d1_RICHEDIT1
7      win_STATIC1
8      win_RADIO1
9      win_RADIO2
10 ENENUM
11 |
12 CREATEDIALOG d1,0,0,400,344,@MINBOX|@MAXBOX|@SIZE|@CAPTION|@SYSTEMENU,0,"Demo Form",&d1_handler
13 CONTROL d1,@STATUS,"Status Window",0,0,0,0,2
14 CONTROL d1,@SysButton,"Cancel",318,265,60,25,0,win_BUTTON1
15 CONTROL d1,@SysButton,"Submit",247,265,60,25,0,win_BUTTON2
16 CONTROL d1,@RICHEDIT,"RichEdit 1",20,45,362,142,@CTEDITMULTI|@CTEDITAUTOV|@CTEDITAUTOH|@CTEDITAUTOV
17 CONTROL d1,@STATIC,"Blog Text",19,15,60,25,@SS_LEFT,win_STATIC1
18 CONTROL d1,@RADIOBUTTON,"On-Line",21,200,132,25,0,win_RADIO1
19 CONTROL d1,@RADIOBUTTON,"Archive",21,224,132,25,0,win_RADIO2
20
21 SHOWDIALOG d1
22 WAITUNTIL ISWINDOWCLOSED(d1)
23 END

```

If the User selects the *Generate skeleton handler* option then the following is the code that is generated.

```

1  DIALOG d1
2  |
3  ▣ ENUM d1_const
4      d1_BUTTON1 = 100
5      d1_BUTTON2
6      d1_RICHEDIT1
7      d1_STATIC1
8      d1_RADIO1
9      d1_RADIO2
10 ENENUM
11 |
12 CREATEDIALOG d1,0,0,400,344,@MINBOX|@MAXBOX|@SIZE|@CAPTION|@SYSTEMENU,0,"Demo Form",&d1_handler
13 CONTROL d1,@STATUS,"Status Window",0,0,0,0,2
14 CONTROL d1,@SysButton,"Cancel",318,265,60,25,0,d1_BUTTON1
15 CONTROL d1,@SysButton,"Submit",247,265,60,25,0,d1_BUTTON2
16 CONTROL d1,@RICHEDIT,"RichEdit 1",20,45,362,142,@CTEDITMULTI|@CTEDITAUTOV|@CTEDITAUTOH|@CTEDITAUTOV
17 CONTROL d1,@STATIC,"Blog Text",19,15,60,25,@SS_LEFT,d1_STATIC1
18 CONTROL d1,@RADIOBUTTON,"On-Line",21,200,132,25,0,d1_RADIO1
19 CONTROL d1,@RADIOBUTTON,"Archive",21,224,132,25,0,d1_RADIO2
20
21 SHOWDIALOG d1
22 WAITUNTIL ISWINDOWCLOSED(d1)
23 END

```

```

24
25 SUB d1_handler(), INT
26   SELECT @MESSAGE
27     CASE @IDINITDIALOG
28       BEGINMENU d1
29         MENUTITLE "&File"
30       ENDMENU
31       CENTERWINDOW d1
32       /* INITIALIZE ANY CONTROLS HERE */
33     CASE @IDCONTROL
34       SELECT @CONTROLID
35         CASE d1_BUTTON1
36           IF @NOTIFYCODE = 0
37             '*BUTTON CLICKED*'
38           ENDIF
39         CASE d1_BUTTON2
40           IF @NOTIFYCODE = 0
41             '*BUTTON CLICKED*'
42           ENDIF
43         CASE d1_RICHEDIT1
44           '* RESPOND TO CONTROL NOTIFICATIONS HERE *'
45         CASE d1_RADIO1
46           IF @NOTIFYCODE = 0
47             '*BUTTON CLICKED*'
48           ENDIF
49         CASE d1_RADIO2
50           IF @NOTIFYCODE = 0
51             '*BUTTON CLICKED*'
52           ENDIF
53       ENDSELECT
54     CASE @IDSIZE
55       CONTROLCMD d1, 2, @SWRESIZE
56     CASE @IDCLOSEWINDOW
57       CLOSEDIALOG d1,@IDOK
58   ENDSELECT
59   RETURN 0
60 END SUB

```

Which ever options the User selects will be saved between sessions of the *Form Editor*.

Saving a Form

Once the current form's layout is as desired, the User can save the form to a file by closing the *Form Editor*. Each time the Form Editor is closed the User is prompted to save any possible pending changes.

Responding in the affirmative results in one of two actions:

1. If the current form is new the User will be presented with a Save As dialog with an automatically suggested file name and default folder. The default folder is the last folder that was used by the *Form Editor*. The User may accept the provided file name and path, or, change either or both. The User can then either save the file or cancel the process. Either will result in the *Form Editor* being closed.
2. If the current form is an existing form the current version will automatically overwrite the old version and the *Form Editor* will close.

Responding in the negative results in the *Form Editor* closing without any saves.

If the User opts to cancel the operation the *Form Editor* will remain open for editing.

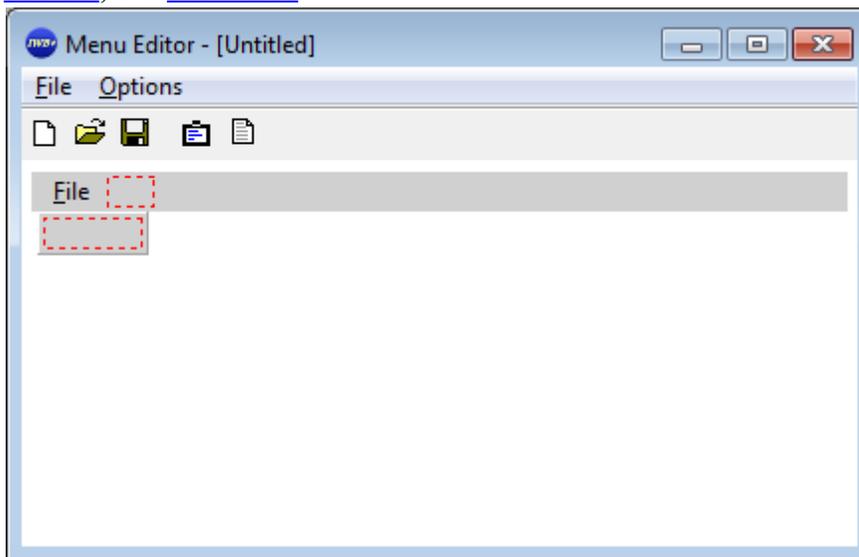
6.3 Menu Editor

Introduction

The *Menu Editor* is a self-contained editor for creating menus for windows (top level only) and dialogs that can later be chosen for use by one or more forms in a project. The *Menu Editor* allows the user to build menus quickly without the worry of typing all the required syntax. However, the user should have a good understanding of how menus are coded for use with IWBASIC. To that end, the user should read the [Windows Programming»Creating and Using Menus](#) section of the Creative Basic Help file before using the Menu Editor.

The *Menu Editor* is opened by selecting the *Tools/Menu Editor* option from [Main Menu](#). When the *Menu Editor* is opened it defaults to a new menu with **File** as the first entry.

The *Menu Editor* User interface is shown below. It consists of a [Caption Bar](#), [Main Menu](#), [Main Toolbar](#), and [Work area](#).



Caption



The Caption Bar always contains the IWBASIC small icon and "Menu Editor".

When a never-saved menu is currently loaded in the *Menu Editor* the caption will also contain the word "Untitled", as shown above.

When a menu source file (.mnu) is loaded the file name will be displayed instead of "Untitled".

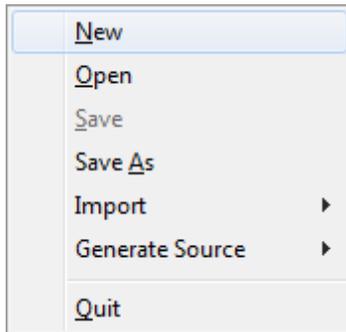
If the menu has unsaved changes pending the file name will be followed by an asterisk "[Untitled]*".

Main Menu



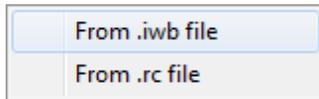
The *Main Menu* bar has 2 options: [File](#) and [Options](#). Each Main Menu option is covered in the sub-sections that follow.

File



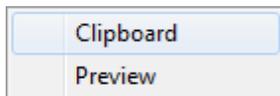
Clicking on *File* results in the dropdown menu shown at left. Each available option (and sub-option) is described below.

Option	Description
New	Creates a new menu in the <i>Menu Editor</i> . If there are any un-saved changes in the current menu the User is prompted to save them.
Open	Opens the <i>Open File</i> dialog to allow the User to reload an existing menu file (.mnu). If there are any un-saved changes in the current menu the User is prompted to save them.
Save	Saves pending changes to an existing menu file. Disabled when current menu has never been saved to a file.
Save As	Opens the Save File As dialog allowing the User to save the current menu to a file.
Import	Opens the <i>Import</i> sub-menu shown below
Generate Source	Opens the <i>Generate Source</i> sub-menu shown below .
Quit	Closes the <i>Menu Editor</i> . If there are any un-saved changes in the current menu the User is prompted to save them.

Import sub-menu

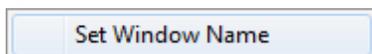
Each available option is described below.

Option	Description
From .iwb file	Opens a dialog used to load a menu from an .iwb source file. If there is no menu in the file an alert will appear. If there are more than one menu definitions in the file the first one encountered while reading the file will be the one loaded.
From .rc file	Opens a dialog used to load a menu from a .rc resource file. If there are more than one menu definitions in the file the first one encountered while reading the file will be the one loaded. Note: The menu definition has to be the first definition in the rc file that contains a BEGIN/END block; otherwise trash may be loaded in the <i>Menu Editor</i> . Also, the ID for any definition in the rc file can not contain the string 'MENU'.

Generate Source sub-menu

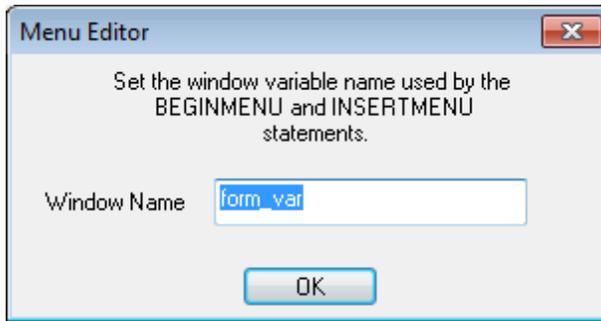
Each available option is described below.

Option	Description
Clipboard	Generates the IWBasic code necessary to create the current menu and copies it to the clipboard. The User can then paste it into an application.
Preview	Generates the IWBasic code necessary to create the current menu and copies it to a preview window which is opened.

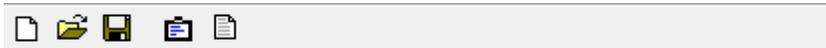
Options

This option is described below.

Option	Description
Set Window Name	Opens a dialog so the User can set the variable name of the window/dialog that the menu will be used in. The default placeholder is 'form_var', as shown below.



Main Toolbar



The main tool bar has 5 buttons. The function of each is described below.

Option	Description
	Creates a new menu in the <i>Menu Editor</i> . If there are any un-saved changes in the current menu the User is prompted to save them.
	Opens the <i>Open File</i> dialog to allow the User to reload an existing menu file (.mnu). If there are any un-saved changes in the current menu the User is prompted to save them.
	Saves pending changes to an existing menu file. If the existing menu has never been saved to a file then the <i>Save File As</i> dialog is opened allowing the User to select a filename. After performing this action the Caption Bar will be updated.
	Generates the IWBASIC code necessary to create the current menu and copies it to the clipboard. The User can then paste it into an application.
	Generates the IWBASIC code necessary to create the current menu and copies it to a preview window which is opened.

Work Area



The *Work Area* is where the desired menu is constructed. When the *Menu Editor* is opened a default new menu with **File** as the first entry is added to the *Work Area*, as shown above.

Getting Started

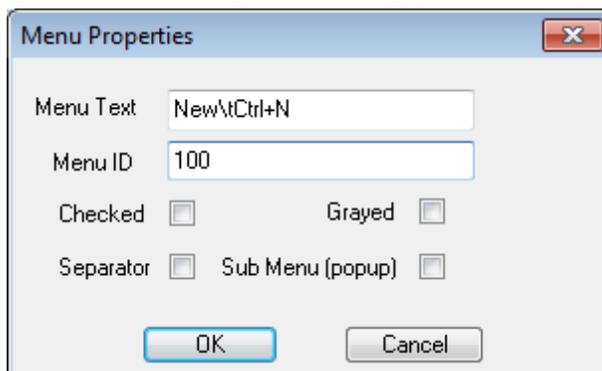
When the *Menu Editor* is opened the User has four options:

- Edit the new menu,
- Open a previously saved menu for editing,
- Import an existing menu that currently resides in a resource file (*.rc) or IWBASIC source file (*.iwb), or
- Exit the program.

Editing

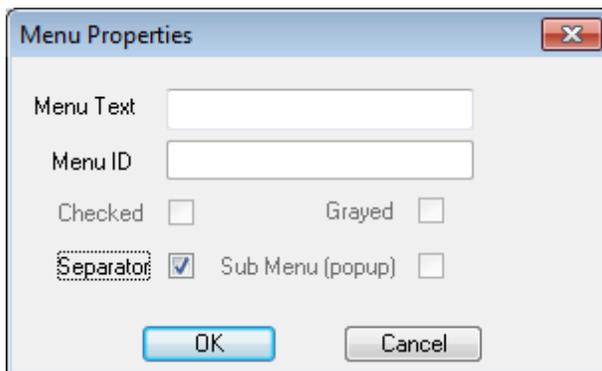
Editing is a very straight forward process. Initially, for a new menu, there is nothing selected. The default title for the first menu column is shown, along with dashed, red rectangles indicating where new items may be added. The [Work Area](#) above shows the default new menu.

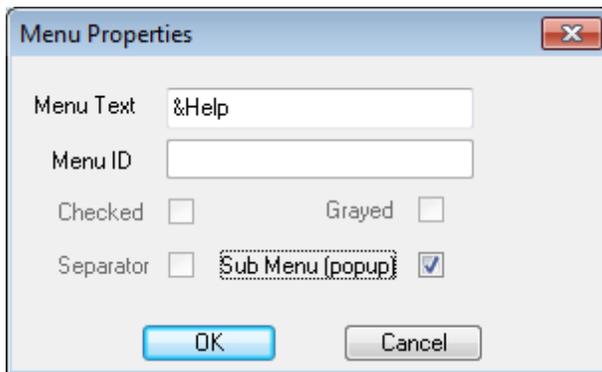
Any shown entry (in the beginning only the default item and the two rectangles) may be selected by left clicking the item. When an item is selected it will have a shaded border. To the bottom and/or to the right are dashed rectangles indicating where new items may be added. To edit the selected item the User can either double-click the item or right-click and select the Properties option. That will open the *Menu Properties* dialog (shown below).



Each element is described in the table below.

Option	Description
Menu Text	The text as it will appear in the final menu. In the example above the \t will cause a tab space to appear in the final menu. Placing a & in front of any letter will cause the letter to be underlined in the final menu when the <ALT> key is pressed. Entry is left blank when the <i>Separator</i> box is checked.
Menu ID	Contains either a unique literal number (as shown above) or a unique User defined constant that equates to a unique number. Left blank when either the <i>Separator</i> or <i>Sub Menu</i> box is checked.
Checked	When checked, will cause the state of the menu item to be initialized in the "checked" state with a corresponding check mark appearing to the left of the menu item in the final menu. Left blank when either the <i>Separator</i> or <i>Sub Menu</i> box is checked.
Grayed	When checked causes the menu item to be initialized in the disabled state. Left blank when either the <i>Separator</i> or <i>Sub Menu</i> box is checked.
Separator	When checked, inserts a separator bar in the menu. When this option is used all other entries in the dialog remain blank as shown in the first example below.
Sub Menu (popup)	When checked, creates a 'parent' menu item which, when clicked, opens a popup menu with additional options. The only other dialog entry that is used is the <i>Menu Text</i> , as shown in the second example below.
OK	The <i>Menu Properties</i> dialog will close and the entry will appear in the displayed menu. The location of the dashed rectangles may or may not change as a result.
Cancel	Closes the <i>Menu Properties</i> dialog without implementing any changes.

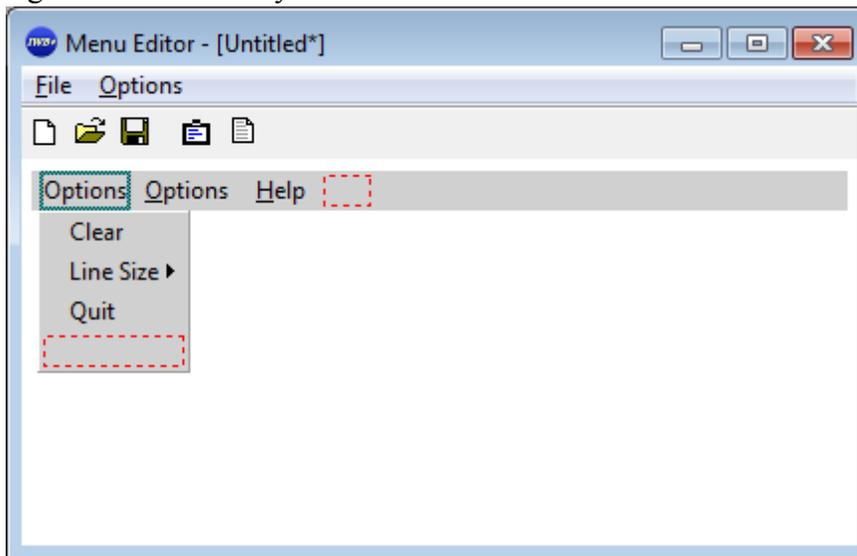




Deleting / Moving Entries

To delete an entry simply right-click on the item and select *Delete* from the popup menu. In the example below right-clicking on the selected entry and clicking *Delete* will remove all entries in that column since it is a top level menu item and has the *Sub Menu* checkbox checked. Once it is deleted it can not be undeleted.

The *Line Size* entry also has a sub menu. Deleting that entry will also delete all of its sub-menus regardless of how many levels of nested sub-menus there are.



Moving an entry is simply a matter of right-clicking the desired entry and selecting the desired Move command.

The available commands are *Move Up*, *Move Down*, *Move Left*, and *Move Right*.

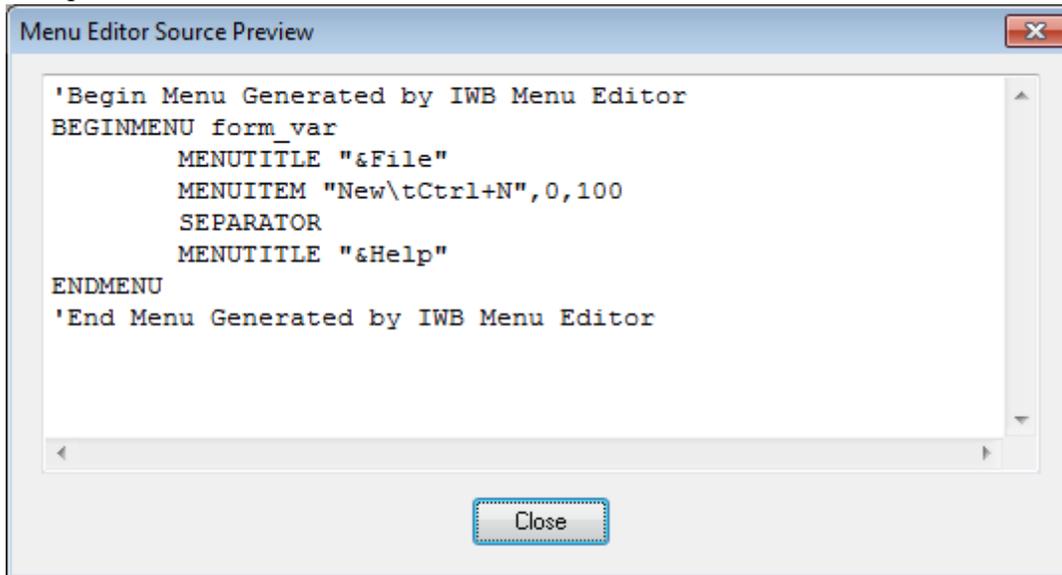
Move Left, and *Move Right* are applicable only to the top level menu entries. Likewise, *Move Up* and *Move Down* only apply to non top level entries.

If the selected entry is at an extreme limit then the corresponding command for that direction will be disabled.

Preview

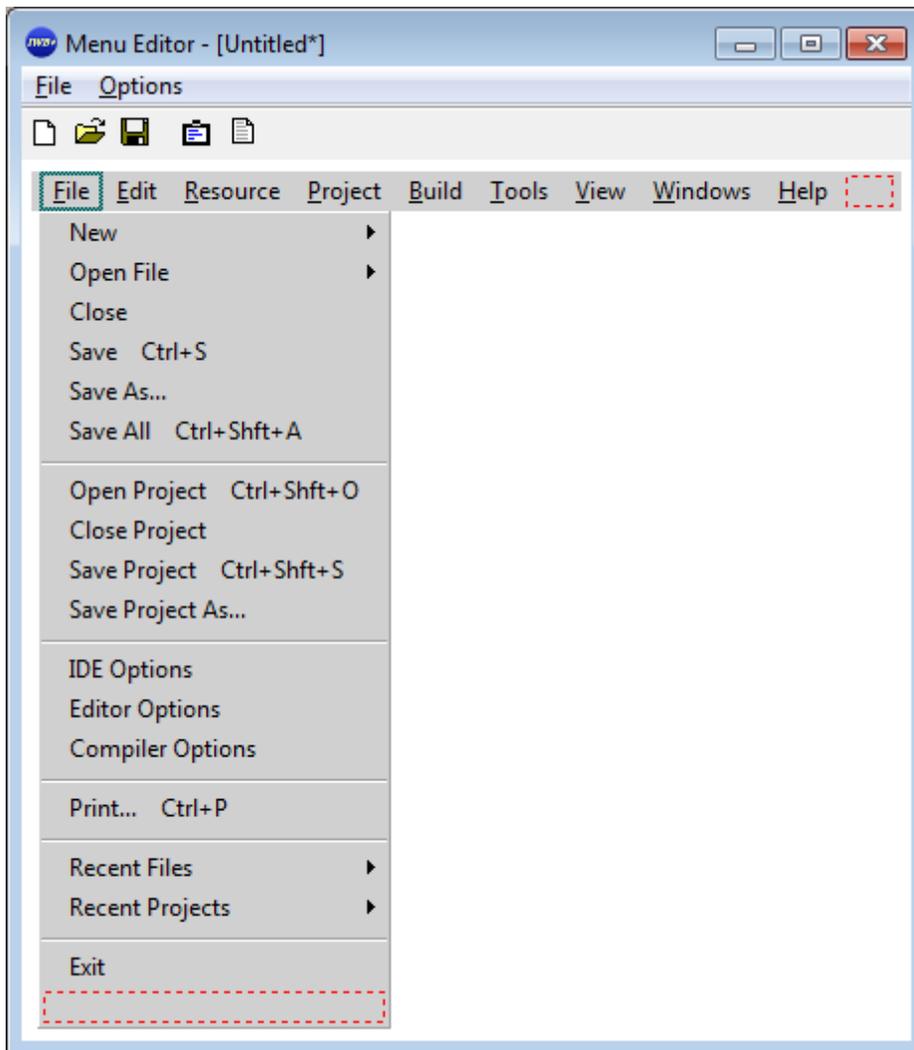
At any time during the process of designing a menu the User can preview the generated code via the

Main Menu Generate Source / Preview option or the Main Toolbar  button. The following is an example.



```
'Begin Menu Generated by IWB Menu Editor
BEGINMENU form_var
    MENUTITLE "&File"
    MENUITEM "New\tCtrl+N",0,100
    SEPARATOR
    MENUTITLE "&Help"
ENDMENU
'End Menu Generated by IWB Menu Editor
```

Finally, the following is the main menu for the IWBASIC IDE that has been imported from the .rc file. The User can compare what this looks like with the appearance of the IDE menu.



6.4 Create Import Library

IWBASIC supports using external DLL's to extend the capabilities of the language through the use of import libraries. An import library is a special .LIB file that contains the names and entry points of all of the functions contained within a particular DLL.

To create an import library for a DLL select *Tools / Create Import Library* in the [Main Menu](#) . A dialog will open where the User can browse to the desired DLL and then click on the *Open* button. The new import library file will be created and have a .lib extension with the same name as the DLL. The file will be copied automatically to the *libs* directory. This only needs to be done once for a new DLL. Once the import library is added it can be used in any program.

For more information see the [Language»Using DLL's and the Windows API](#) section.

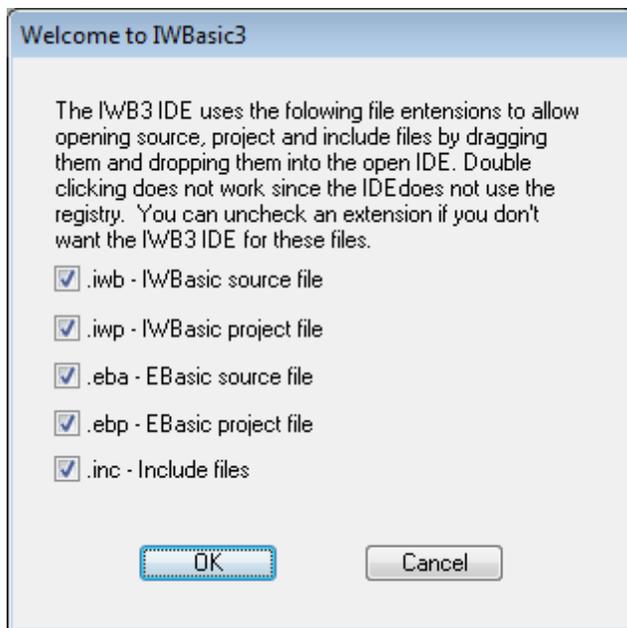
6.5 Register File Extensions

When the User installs an application on their own computer it is usually desirable to associate certain file extensions with that application. Doing so allows the User to double-click a file in Windows Explorer and have the application automatically open and then load the file. However, since IWBASIC does not use the registry, there is no way to automatically open the IDE by double-clicking a file. The User can optionally associate certain file extensions with the IDE.

This association will do two things:

- It will assign a unique icon to each file type in Windows Explorer
- It will allow the User to drag-and-drop that file type into an open instant of the IWB3 IDE for editing.

Clicking *Tools/Register File Extensions* from the [Main Menu](#) opens the following dialog:



The User selects the file extensions to register.

Selecting *OK* will make the required entries in the computer's registry and close the dialog.

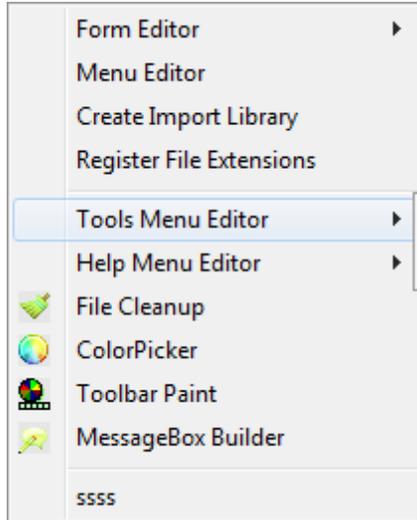
Selecting *Cancel* will close the dialog without making any changes.

Note: Registering file extensions is an operating system function. As such, it should only be used on the User's personal computer. This is the only registry modifications that IWBASIC makes under any circumstances.

WARNING: Making these associations will remove any and all previous associations made with previous versions.

6.6 Tools Menu Editor

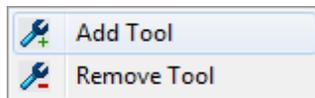
Introduction



IWBASIC comes with ten utility tools as an integral part of the application. Those ten are the first ten options in the *Tools* option of the [Main Menu](#) (shown at left). They are all covered in detail in the [Utilities](#) section.

However, each User has the ability to add their own personal favorite tools to the *Tools* menu. An example is shown by the *ssss* menu option, at left, below the separator line.

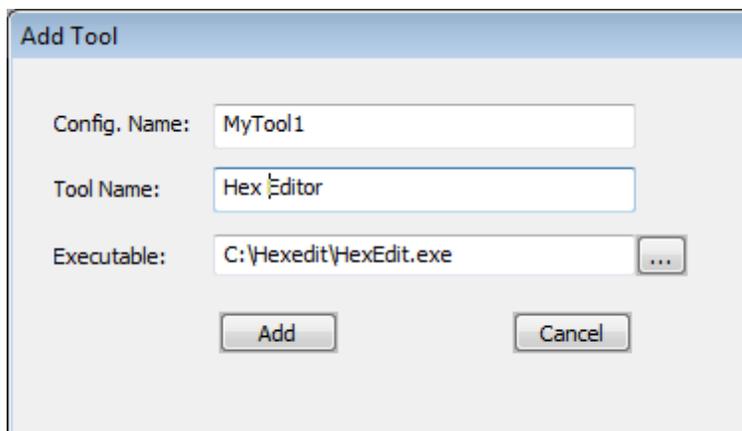
Note: There is no internal limit on the number of tools that can be added to the *Tools* menu. The only limit is how many tools will fit (and be viewable) on the User's screen.



The *Tools / Tools Menu Editor* option allows the User to add/delete User tools to/from the *Tools Menu* via the *Tools Menu Editor* sub-menu.

Adding Tools

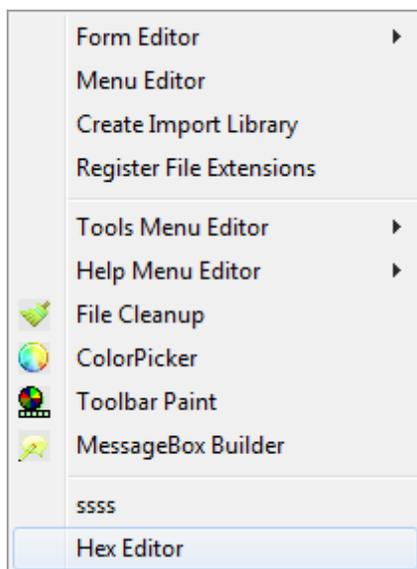
To add a tool, the User selects the *Tools / Tools Menu Editor / Add Tool* option from the [Main Menu](#).



The *Tool Menu Editor's Add Tool* dialog contains three entries and three buttons, discussed below.

The example at left contains typical entries

Item	Description
Config. Name	This is the unique name of the file (with '.inct' extension) that will contain this tool's information. All '*.inct' files are located in the User's <i>My Documents/Application Data</i> folder for IW BASIC for a normal installation and in the <i>IWBDec/tools</i> folder for a "memory stick" installation..
Tool Name	This is the tool's name as it will appear in the Tools menu.
Executable	This is the full path name to the tool's executable file.
	Opens a standard Open File dialog to facilitate the entry of the executable file path/name.
Add	Validates the entries, creates the *.inct file, and rebuilds the <i>Tools</i> menu so the new addition is immediately available.
Cancel	Aborts the addition.

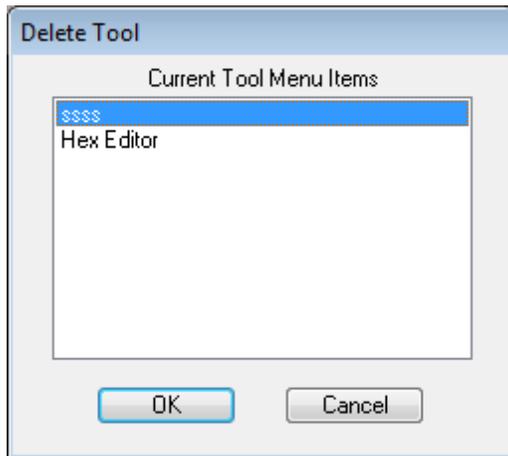


With the above *Add Tool* dialog entries as an example, the *Tools* menu would now appear as shown.

Deleting Tools

To delete a tool, the User selects the *Tools / Tools Menu Editor / Delete Tool* option from the

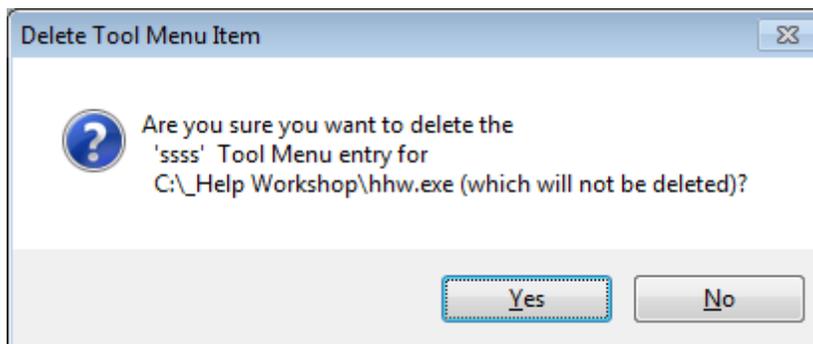
Main Menu.



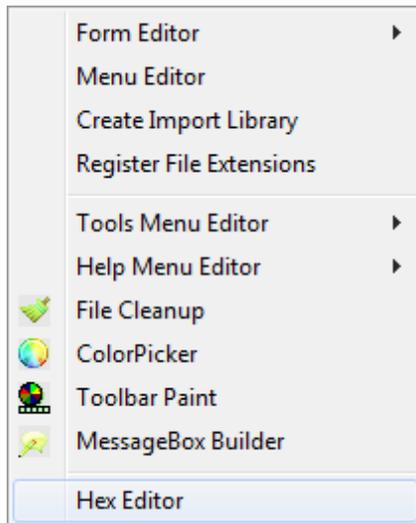
The *Tools Menu Editor's Delete Tool* dialog appears with all User added tools listed. The User selects one or more tools to delete. Click *Cancel* to abort the process.

Clicking the *OK* button:

- The User is prompted to confirm each deletion (shown below).
- The associated *.inct is deleted from the haddisk.
- The associated executable file is NOT deleted.
- The option is removed from the *Tools* menu.



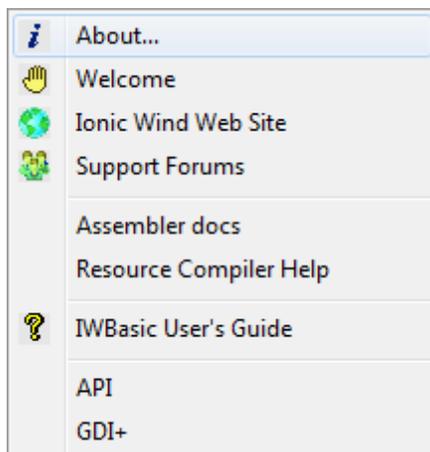
Click *Yes* to continue with the deletion of this tool. Click *No* to abort the deletion of this tool.



Clicking *Yes* with the above example results in the *Tools* menu, as shown.

6.7 Help Menu Editor

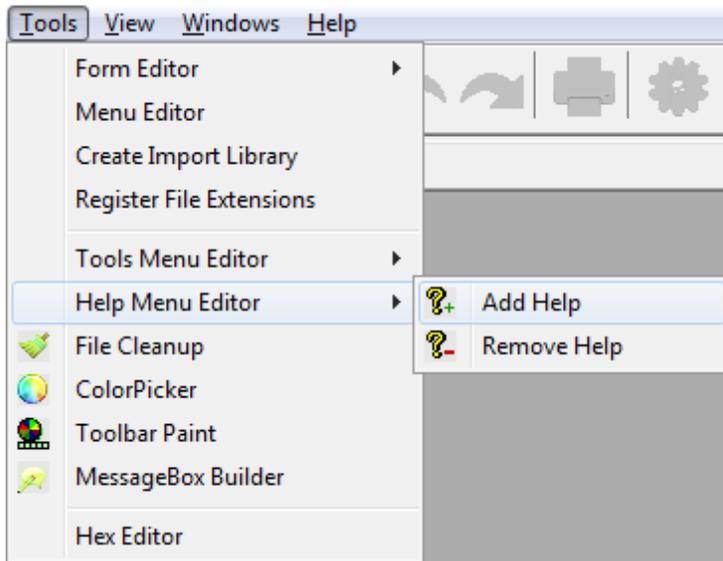
Introduction



IWBASIC comes with seven pre-configured *Help* menu options as an integral part of the application. Those seven are the first seven options in the *Help* menu (shown at left). They are covered in detail in the [Main Menu](#) section.

However, each User has the ability to add other help files of their choosing to the *Help* menu. An example is indicated by the two options, at left, below the separator line.

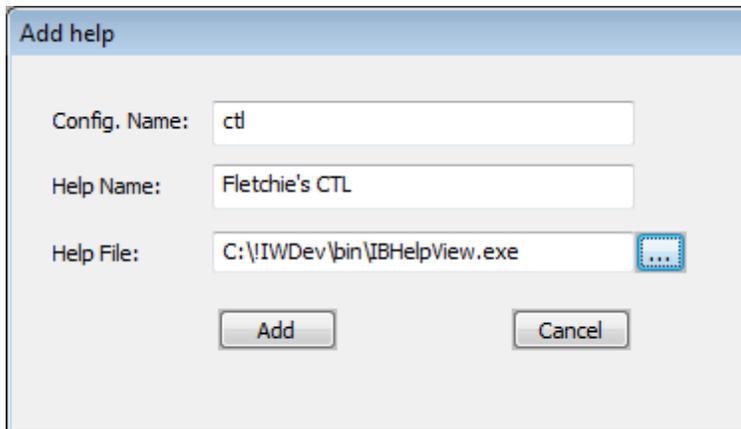
Note: There is no internal limit on the number of help files that can be added to the Help menu. The only limit is how many entries will fit (and be viewable) on the User's screen.



The *Tools / Help Menu Editor* option allows the User to add/delete User help files to/from the *Help Menu* via the *Help Menu Editor* sub-menu.

Adding Help Files

To add a help file, the User selects the *Tools / Help Menu Editor / Add Tool* option from the [Main Menu](#).

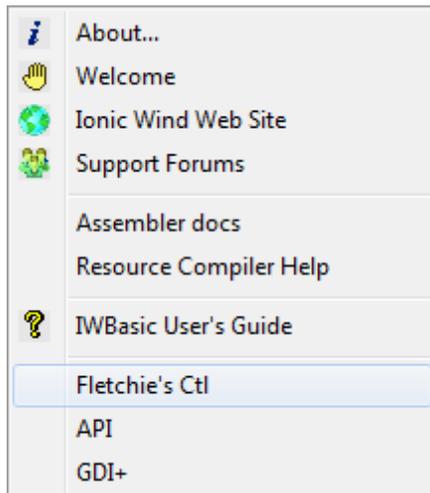


The *Help Menu Editor's Add Help* dialog contains three entries and three buttons, discussed below.

The example at left contains typical entries

Item	Description
Config. Name	This is the unique name of the file (with '.incc' extension) that will contain this help's information. All '*.incc' files are located in the User's <i>My Documents/Application Data</i> folder for IW BASIC for a normal installation and in the <i>IWBDec/help2</i> folder for a "memory stick" installation.

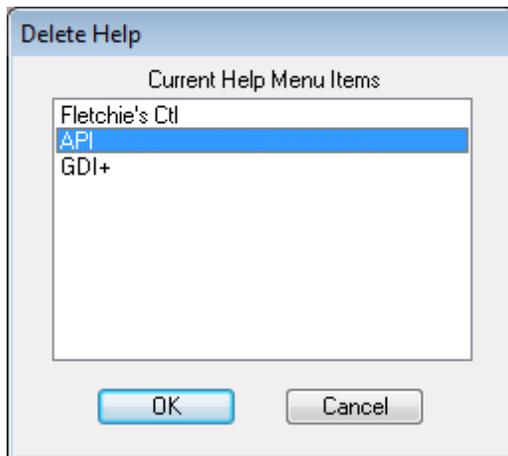
Tool Name	This is the help file's name as it will appear in the <i>Help</i> menu.
Executable	This is the full path name to the help file. NOTE: The default is for *.chm files. However, IWBASIC does allow the User to select *.pdf and EBook (*.exe) files.
	Opens a standard <i>Open File</i> dialog to facilitate the entry of the executable file path/name.
Add	Validates the entries, creates the *.incc file, and rebuilds the <i>Help</i> menu so the new addition is immediately available.
Cancel	Aborts the addition.



With the above *Add Help* dialog entries as an example, the Help menu would now appear as shown.

Deleting Help Files

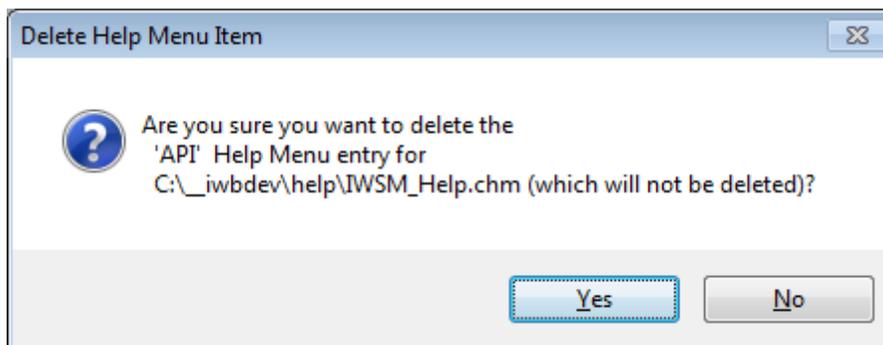
To delete a help file, the User selects the *Tools / Help Menu Editor / Delete Help* option from the [Main Menu](#).



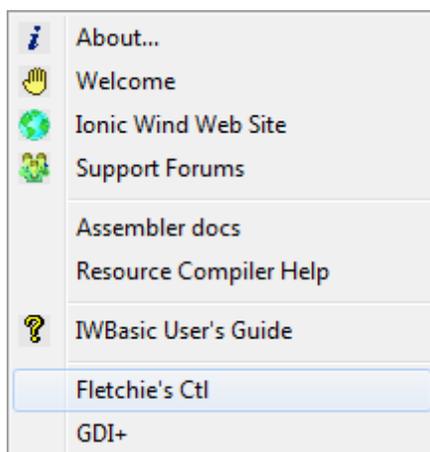
The *Help Menu Editor's Delete Help* dialog appears with all User added help files listed. The User selects one or more help files to delete. Click *Cancel* to abort the process.

Clicking the *OK* button:

- The User is prompted to confirm each deletion (shown below).
- The associated *.incc is deleted from the harddisk.
- The associated executable file is NOT deleted.
- The option is removed from the *Help* menu.



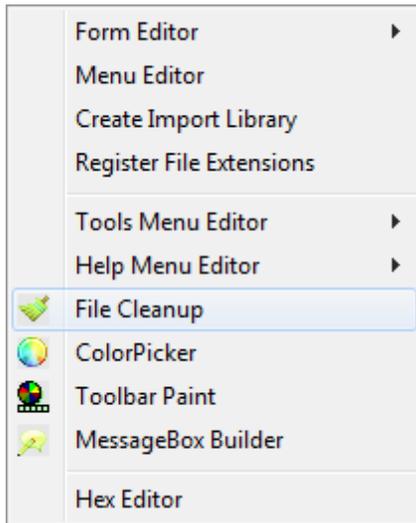
Click *Yes* to continue with the deletion of this help file. Click *No* to abort the deletion of this help file.



Clicking *Yes* with the above example results in the *Help* menu, as shown.

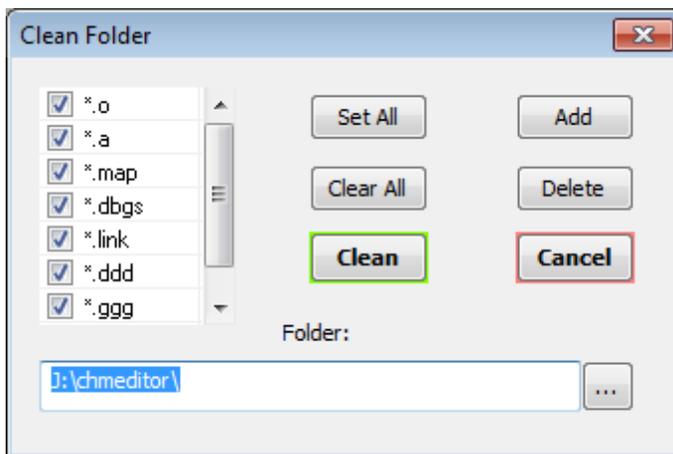
6.8 File Cleanup

When compiling/linking/debugging IWBASIC applications there are intermediate files that are created. These files are no longer needed when work on the application has been completed. The *File Cleanup* utility allows the User to easily remove these files.



To use this utility, the User selects *Tools / File Cleanup* option from the [Main Menu](#).

This opens the *Clean Folder* dialog (shown below).



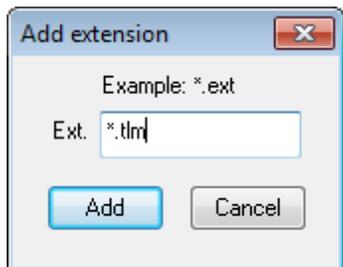
The *Clean Folder* dialog contains a List box, seven buttons, and an edit control. The function of each is described in the table below.

Item	Description
-	List box with a list of file extensions. Each entry has an associated checkbox. When a box is checked that file extension will be used in the cleansing process.
Set All	Checks all the check boxes
Clear All	Clears all the check boxes.
Clean	When clicked all files in the specified folder that have an extension that is checked in the list box will be deleted. This action is non-reversible. When completed the dialog is closed.

Add	Opens the <i>Add extension</i> dialog, shown below.
Delete	Removes the selected extensions from the list box. See additional details below.
Cancel	Closes the dialog.
Folder	Contains the full path name to the folder to be cleaned.
	Opens the <i>Select Folder</i> dialog to allow the User to select a folder to clean.

Adding Extension

Clicking the *Add* button opens the *Add extension* dialog.

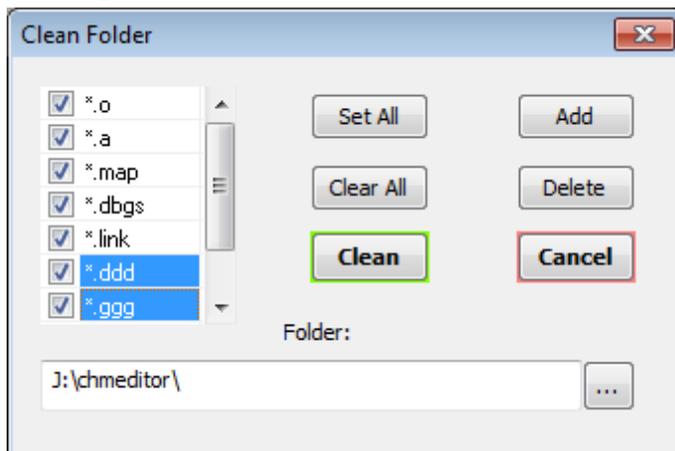


Enter the desired extension in the format shown.

Clicking *Add* will add the extension to the list box in the *Clean Folder* dialog shown above.

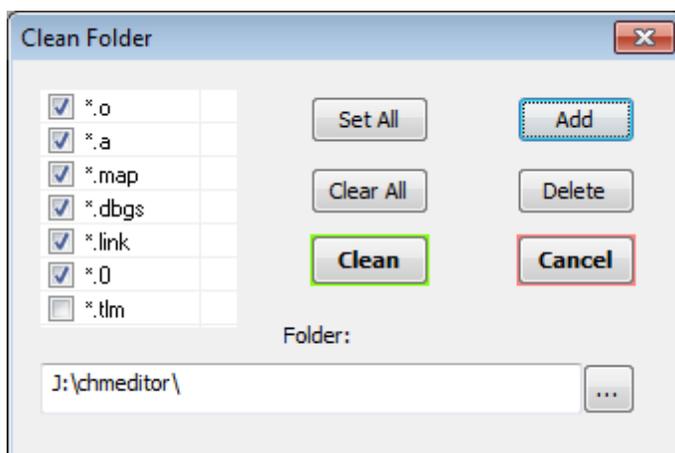
Clicking *Cancel* closes the dialog without taking any action.

Deleting Extensions



To delete one or more extensions select them in the list box and then click *Delete*. The entries will immediately be removed from the list box.

Note: The first five entries are considered default entries and can not be removed from the list box. Any attempt to do so will be ignored.

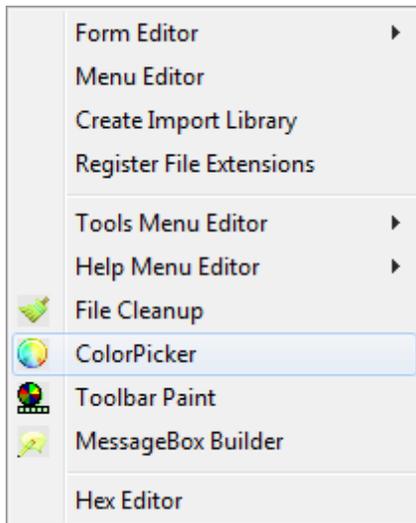


The add and delete activities described above result in the list box entries shown at left.

6.9 ColorPicker

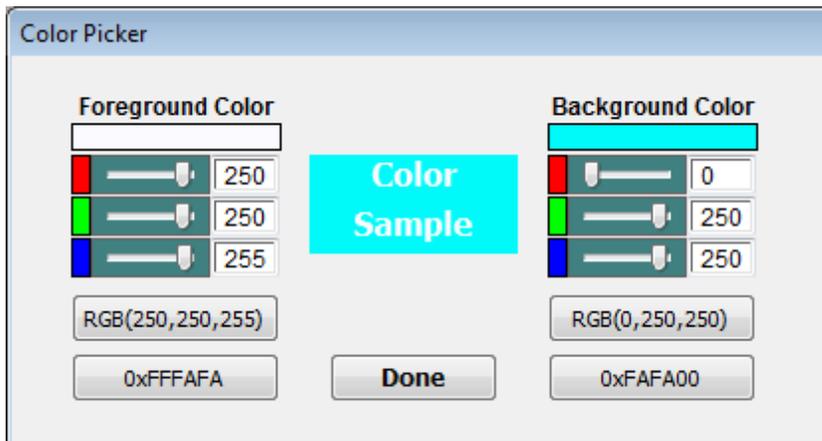
There are times when the User needs to select a foreground and/or background color for a window/control. There are also times when the User needs to see the combination of colors together in order to make the proper selection.

The *ColorPicker* utility is designed to facilitate those decisions.



To use this utility, the User selects *Tools / ColorPicker* option from the [Main Menu](#).

This opens the *Color Picker* dialog (shown below).



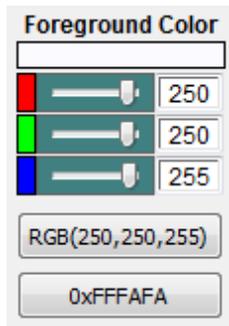
The *Color Picker* dialog consist of three sections:

- A Foreground Color Control Group.
- A Background Color Control Group
- A Preview window.

All three are discussed below.

Clicking *Done* closes the dialog.

Color Control Group



A Color Control Group is shown at left.

The following information applies equally to either the Foreground or Background colors.

Item	Description
Title	The name of the parameter being selected . In this application it will be either Foreground or Background.
Color Bar	Displays the current color of the parameter. Clicking on the bar will open the standard System Color dialog, allowing the User to select a color. If a color is selected in this manner the 3 associated Trackbar Clusters (below) will automatically be updated to reflect the selection.
Trackbar Clusters	Each of the three Trackbar Clusters are compose of three components: <ul style="list-style-type: none"> • A small colored rectangle indicating which color component the trackbar controls (red, green, blue). • The actual trackbar. A standard IWBasic trackbar. See description of operation below. • An edit window whose value tracks the setting of the trackbar. If a value is entered directly then the trackbar will be updated to reflect the change. The range of allowable values is 0-255.
RGB Button	Button text immediately reflects any change made above; in the proper IWBasic RGB format. Clicking the button copies the buttons text to the clipboard. The User may then proceed to paste the text at the desired point.
Hex Button	Button text immediately reflects any change made above; in the proper IWBasic Hex format. Clicking the button copies the buttons text to the clipboard. The User may then proceed to paste the text at the desired point.



The Color Preview Window displays the current *ColorPicker* Foreground or Background colors.

Trackbar Refresher

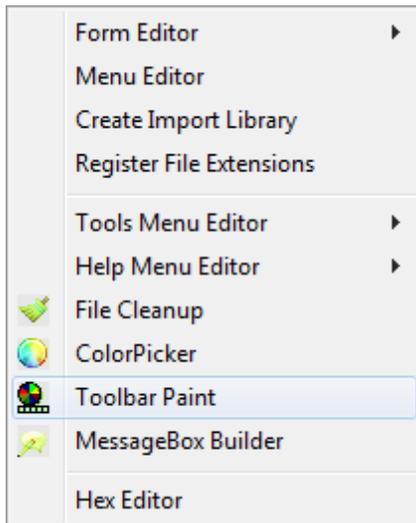
The trackbar's position can be changed in any one of several ways:

- Dragging the pointer with the mouse.
- Clicking to the side of the pointer will inc/dec the value by 1.
- Pressing the left/right arrows keys on the keyboard while the trackbar has focus will inc/dec the value by 1.
- Entering a value directly into the edit control associated with the trackbar.

Any change in trackbar value will result in a corresponding change in its "buddy" edit control.

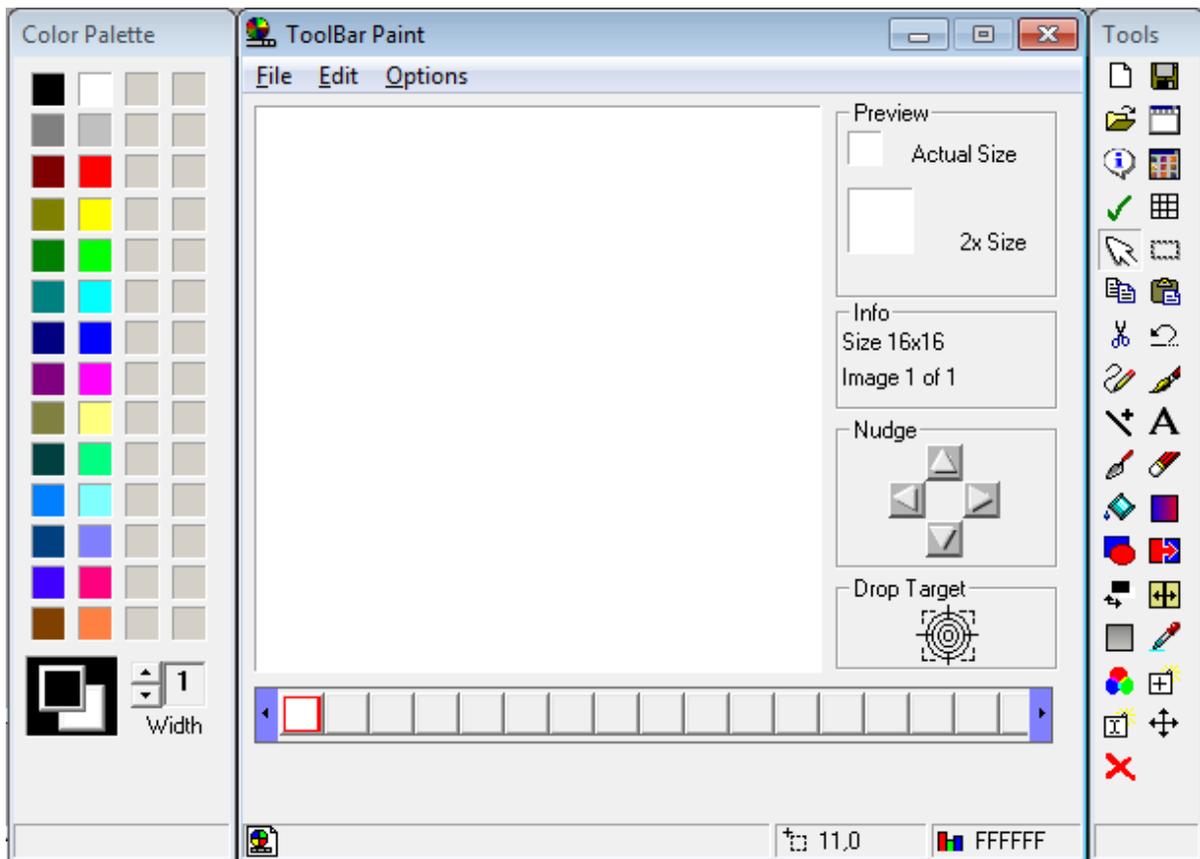
6.10 Toolbar Paint

The *Toolbar Paint*© utility is a 3rd party program created and copyrighted 2003 by Edgar Hanson. It is free to use and distribute per the licensing agreement in the program's Option menu. The creator chose not to create a help file and it is not within the scope of this project to create one either. It is very intuitive to use.



To use this utility, the User selects *Tools / Toolbar Paint* option from the [Main Menu](#).

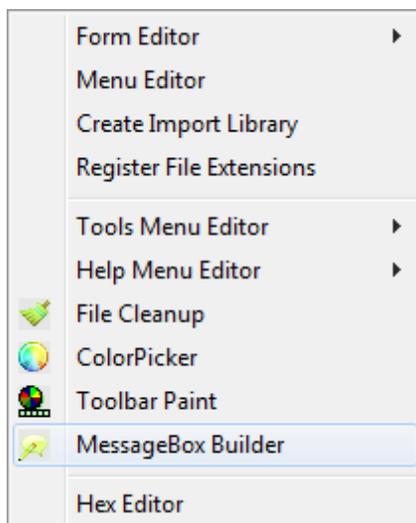
This opens the *Toolbar Paint* program (shown below).



Toolbar bitmaps created with this utility can be used with the `LOADTOOLBAR` command.

6.11 MessageBox Builder

The *MessageBox Builder* utility provides a simple, quick means to create and configure MessageBoxes.



To use this utility, the User selects *Tools / MessageBox* option from the [Main Menu](#).

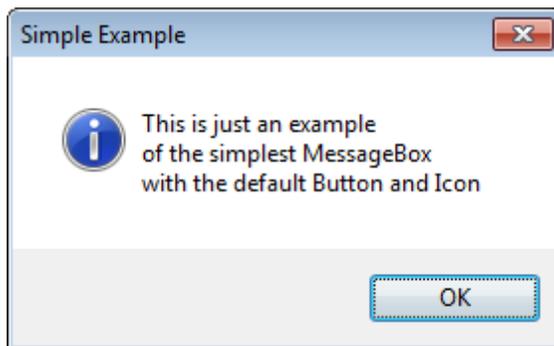
This opens the *MessageBox Builder* dialog (shown below).

The dialog at left contains entries/selections for a simple, multi-line messagebox. The elements of the *MessageBox Builder* dialog are described in the table below.

Item	Description
Win	The variable name of the messagebox's parent window.
Title	The text that will appear in the messagebox's caption.
Msg	The actual text of the message. Trailing spaces are preserved.
Button(s)	Column of mutually exclusive radiobuttons used to select which button combination will appear in the messagebox.
Icon	Column of mutually exclusive radiobuttons used to select which icon will appear in the messagebox.

Save	Automatically generates the appropriate code for the configured messagebox and saves it to the clipboard. Closes the <i>MessageBox Builder</i> dialog
Cancel	Closes the <i>MessageBox Builder</i> dialog without saving anything to the clipboard.
Preview	Displays a preview version of the configured messagebox (as shown below).

Note: All edits made during an instance of the *MessageBox Builder* will remain in place for the next instance. The only time all the entries will be reset is when IW BASIC is restarted.

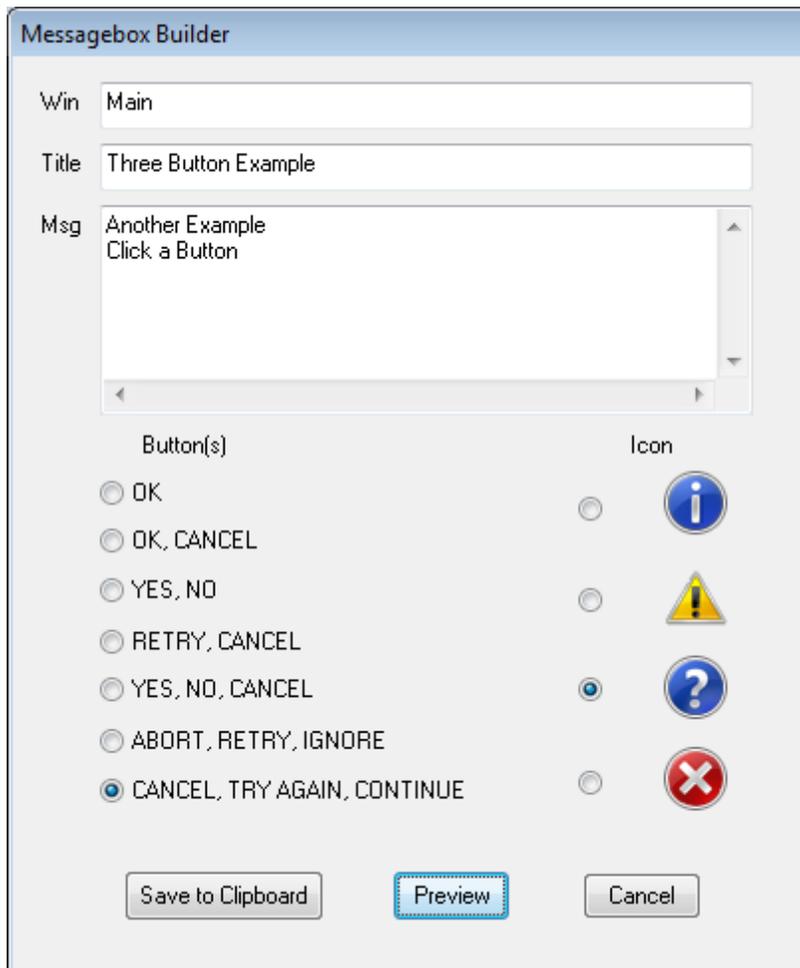


The Preview of the messagebox configured above.

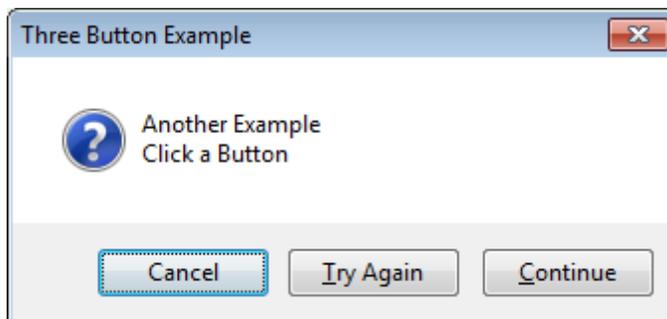
After clicking the *Save* button in the *MessageBox Builder* dialog the User opens a [Code Editor](#) window and places the caret at the desired point of insertion. Any text on that line will automatically be moved down when the User pastes the code from the clipboard.

Below is the inserted code for the simple example.

```
MESSAGEBOX (win, "This is just an example\nof the simplest MessageBox\nwith the default Button and Icon", "Simple Example", @MB_OK|@MB_ICONINFORMATION)
```



This is an example of the entries for a three button messagebox and a non-default icon.



This is the Preview of the above configuration.

And this is the resulting code pasted into the Code Editor.

```

INT msgret = MESSAGEBOX (Main, "Another Example\nClick a Button", "Three Button Example", @MB_CANCELTRYCONTINUE|@MB_ICONQUESTION)
SELECT msgret
CASE @IDCANCEL
CASE @IDRETRY
CASE @IDCONTINUE
ENDSELECT

```

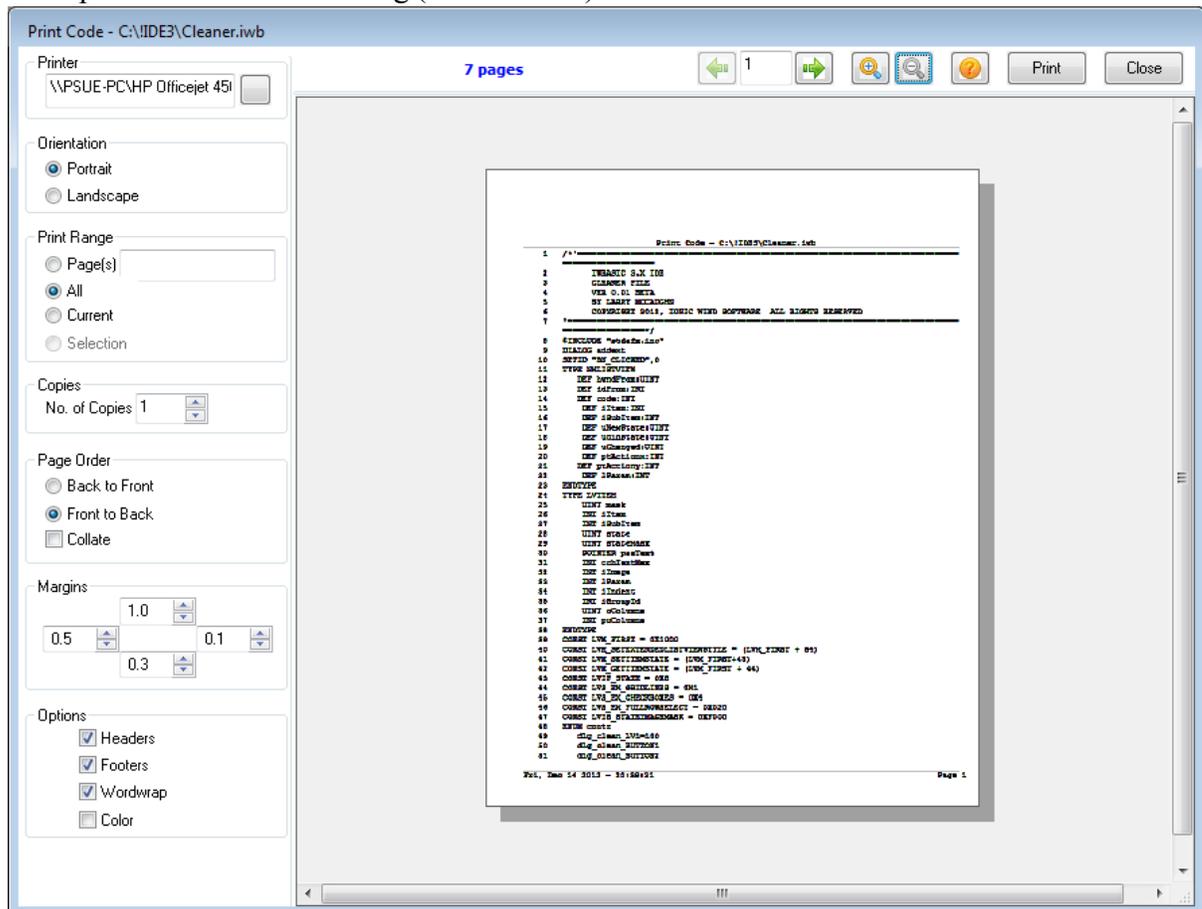
6.12 Print Preview

Introduction

The *Print Preview* utility allows the User to configure how source files, opened in the [Code Editor](#), will appear when all or portions are printed.

To use this utility, the User clicks the  button on the [Main Toolbar](#) or *File /Print* on the [Main Menu](#). The currently selected [Code Editor](#) file in the [Workspace](#) is loaded into the Print Preview. Note: The  button is disabled when there is no currently select [Code Editor](#) window.

This opens the *Print Code* dialog (shown below).



The *Print Code* dialog is divided into ten sections; [Caption](#), [Main Toolbar](#), [Printer](#), [Orientation](#), [Print Range](#), [Copies](#), [Page Order](#), [Margins](#), [Options](#), and [Preview Window](#).

Caption

Print Code - C:\IIDE3\Cleaner.iwb

The *Print Code* window caption contains the full path name of the file currently being processed.

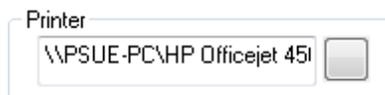
Main Toolbar



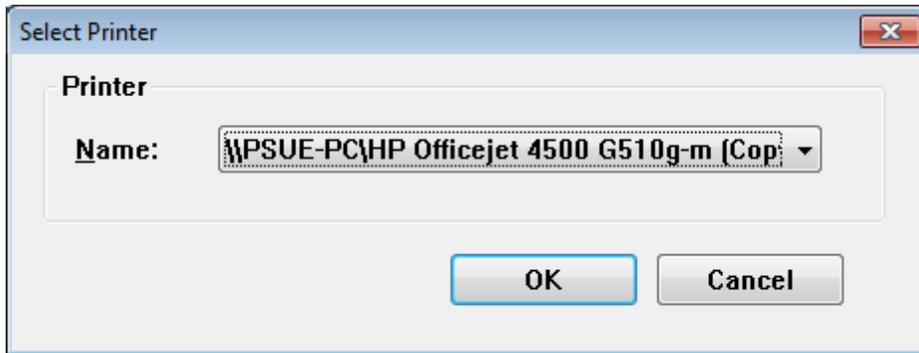
The *Main Toolbar* has 9 items. The function of each is described below.

Item	Description
	Indicates the number of sheets of paper required to print all the text currently being processed, as configured. May change as various options are selected and parameters are changed.
	Used to change the currently displayed preview page, in the <i>Preview Window</i> , to the preceding page. The button is disabled when the lower limit on page numbers is reached.
<input type="text" value="1"/>	Indicates the preview page currently being displayed in the <i>Preview Window</i> . When the preview window is opened it always defaults to the first page
	Used to change the currently displayed preview page, in the <i>Preview Window</i> , to the following page. The button is disabled when the upper limit on page numbers is reached.
	Used to zoom in, in the <i>Preview Window</i> .
	Used to zoom out, in the <i>Preview Window</i> .
	Opens this help file to this section.
<input type="button" value="Print"/>	Used to print the text as it is currently configured in the <i>Code Print</i> dialog.
<input type="button" value="Close"/>	Closes the <i>Code Print</i> dialog.

Printer



Displays the currently selected printer that will be used for any printing. Clicking the button opens a printer dialog (shown below) that lists all available printers. The User may select a different desired printer. The entry here is saved each time the *Print Code* dialog is closed and reloaded when the dialog is reopened.



Orientation

Orientation

Portrait

Landscape

Used to set the page orientation of the printed output. Any time this option is changed the contents of the *Preview Window* will immediately change. Making a change will usually result in the number of pages (discussed in the above table) to change also. This entry is saved each time the *Print Code* dialog is closed and reloaded when the dialog is reopened.

Print Range

Print Range

Page(s)

All

Current

Selection

The User has four options to choose from when printing the current contents of the preview. Each is described in the following table:

Option	Description
Page(s)	This option results in the page numbers in the associated edit window being printed. Individual page numbers and/or a range of page numbers are separated by a comma. A range of page numbers is specified with a starting page and an ending page separated with a ' - '. If the ending page number is less than the starting number or is blank then the total page number (shown in the <i>Main Toolbar</i>) will be used for the ending number. Pages will be printed in numerical order regardless of the order they are entered. If a nonexistent page number is entered it will be ignored. If the User enters numbers that are duplicates the duplicated page will only be printed once. In the following examples it is assumed that there are a max of 12 pages in the

	preview: 1. Given 9,1-2,11- results in pages 1,2,9,11,12 being printed. 2. Given 9,1-2,8-6 results in pages 1,2,8,9,10,11,12 being printed. 3. Given 1-5,6,4-7,3-9 results in pages 1,2,3,4,5,6,7,8,9 being printed This option is disabled when the <i>Selection</i> option is selected.
All	This is the default selection when the <i>Print Code</i> dialog provided there was no text selected in the active Code Editor window when the <i>Print Code</i> dialog was opened. This option results in all pages being printed. This option is disabled when the <i>Selection</i> option is selected.
Current	This option prints only the page currently displayed in the <i>Preview Window</i> . The User can change pages via the arrow buttons in the <i>Main Toolbar</i> . This option is disabled when the <i>Selection</i> option is selected.
Selection	This option prints all pages currently loaded in the <i>Print Code</i> dialog. The option is automatically selected when there is selected text in the active Code Editor window when the <i>Print Code</i> dialog is opened. Only the selected text is loaded into the <i>Print Code</i> dialog. This option is disabled when there is no selected text when the <i>Print Code</i> dialog is opened.

Copies

Copies

No. of Copies

Used to set how many times each page will be printed. Defaults to one copy of each page.

Page Order

Page Order

Back to Front

Front to Back

Collate

This section is used to establish the order that multiple pages. The first two options are used to compensate for how the User's print handles page printing. The ultimate goal is to have the pages printed in order with the first page the looks at is the first page of the text. The easiest way to determine which option to pick is to look at the way the printed page comes out of the User's printer. If the printed page comes out with the printed side down then the proper selection would be *Front to Back*. If the printed page comes out with the printed side up then the proper selection would be *Back to Front*.

Collate is only applicable when the User is printing multiple copies of multiple pages. As an example, let's assume the User wants to print pages 1,2,3 and wants 3 copies of each.

With the *Front to Back* option:

If the *Collate* option is not checked the pages would be printed in the following order:

1,1,1,2,2,2,3,3,3.

If the *Collate* option is checked the pages would be printed in the following order:

1,2,3,1,2,3,1,2,3.

With the *Back to Front* option:

If the *Collate* option is not checked the pages would be printed in the following order:

3,3,3,2,2,2,1,1,1.

If the *Collate* option is checked the pages would be printed in the following order:

3,2,1,3,2,1,3,2,1.

This entry is saved each time the *Print Code* dialog is closed and reloaded when the dialog is reopened.

Margins

Margins

1.0

0.5 0.1

0.3

The *Margins* section contains four edit fields with associated up/down controls. Their relative layout location indicates which margin each controls. The up/down controls change the margins in 1/10th of an inch intervals. As the User changes a given margin the change is automatically indicated in the *Preview Window*. Reducing all margins to 0.0 does not necessarily allow printing at the absolute edge of the paper. Most printers have a small dead area around the edge of the paper. The *Preview Window* takes that value into account in its calculations.

These entries are saved each time the *Print Code* dialog is closed and reloaded when the dialog is reopened.

Options

Options

Headers

Footers

Wordwrap

Color

The final four Options are described in the following table:

Option	Description
Headers	The header contains the name of the file. If checked the header will be printed adjacent to the top margin. If it is not checked the space it would have used will

	be given back for printing the file text. The header is followed by a line which will always be there regardless of whether or not the header is printed.
Footers	The footer contains the date and time the page was printed on the left side and the page number on the right side. If checked the footer will be printed adjacent to the bottom margin. If it is not checked the space it would have used will be given back for printing the file text. The footer is preceded by a line which will always be there regardless of whether or not the footer is printed.
Wordwrap	When checked, lines that are longer than the allowable print width are properly wrapped so that no characters are lost. Wrapped lines are easy to spot since they will appear as a line with no line number. If the option is not checked then long lines will be truncated and text will be lost.
Color	When checked the printed pages will appear exactly the way they do in the IDE's Code Editor windows. Otherwise, all printing will be in black on white.

The above entries are saved each time the *Print Code* dialog is closed and reloaded when the dialog is reopened.

Preview Window

The *Preview Window* displays the text as it will appear when printed with all the current configuration settings.

There are 3 levels of zoom. The 1st level shows a preview of the entire page (in the first example below). This is the default each time the *Code Print* dialog is opened.

The 2nd level of zoom (second example below) shows the page zoomed-in to the point that it fills approx. 90% of the available width of the default *Code Print*.dialog size.

The 3rd level of zoom (third example below) shows the page at actual size.

When the preview image exceeds the available viewing area the scrollbars are available so that all portions of the preview may be seen.

```

Print Code - C:\IWS\Cleaner.lsh
1 /*-----
2
3 THEASTIC 8.X IWS
4 CLEANER FILE
5 VER 0.01 BETA
6 BY GARY MCCAGHEE
7 COPYRIGHT 2015, IONIC WIND SOFTWARE ALL RIGHTS RESERVED
8
9 -----*/
10 #INCLUDE "wdsafe.inc"
11 #DEFINE adjust
12 #DEFINE "EM_CHECKED",0
13 TYPE HANDLEVIEW
14 DEF hndForm:INT
15 DEF idFrom:INT
16 DEF code:INT
17 DEF iItem:INT
18 DEF iSubItem:INT
19 DEF uMemProc:UINT
20 DEF uMemProc:UINT
21 DEF uMemProc:UINT
22 DEF pAction:INT
23 DEF iParam:INT
24
25 #STRUCT
26 TYPE LVIEW
27 INT mask
28 INT iItem
29 INT iSubItem
30 INT STATE
31 INT STATEMASK
32 POINTED pntText
33 INT cchTextMax
34 INT iImage
35 INT iParam
36 INT iParamG
37 INT iStrupId
38 INT oColumns
39 INT poColumns
40
41 #DEFINE
42 CONST LVM_FIRST = 0x1000
43 CONST LVM_NOTIFYINDEXSELECTEDSTATE = (LVM_FIRST + 01)
44 CONST LVM_NOTIFYINDEXSTATE = (LVM_FIRST+43)
45 CONST LVM_NOTIFYINDEXSTATE = (LVM_FIRST + 44)
46 CONST LVM_STATE = 0x0
47 CONST LVM_EM_CHECKED = 0x1
48 CONST LVM_EM_CHECKEDONES = 0x4
49 CONST LVM_EM_FULLSCREEN = 0x800
50 CONST LVM_STATE_FULLSCREEN = 0x1000
51
52 #END
53
54 #if _clean_LVM=0
55 #else
56 #endif
57
58 #if _clean_LVM=0
59 #else
60 #endif
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612

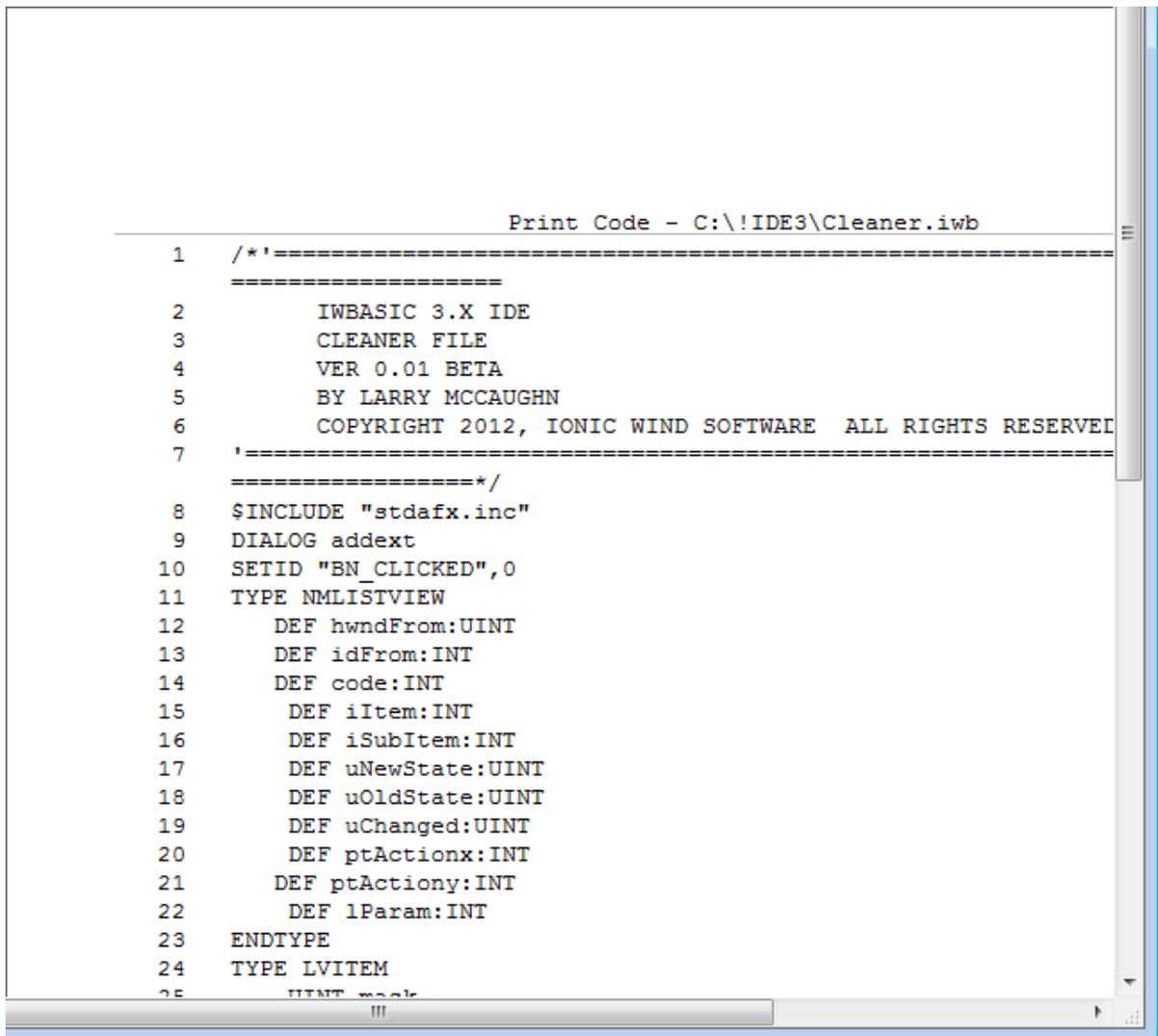
```

```

Print Code - C:\!IDE3\Cleaner.iwb
1  /*=====
2      IWBASIC 3.X IDE
3      CLEANER FILE
4      VER 0.01 BETA
5      BY LARRY MCCRAUGHN
6      COPYRIGHT 2012, IONIC WIND SOFTWARE  ALL RIGHTS RESERVED
7  '=====*/
8  $INCLUDE "stdafx.inc"
9  DIALOG addext
10 SETID "BN_CLICKED",0
11 TYPE NMLISTVIEW
12     DEF hwndFrom:UINT
13     DEF idFrom:INT
14     DEF code:INT
15     DEF iItem:INT
16     DEF iSubItem:INT
17     DEF uNewState:UINT
18     DEF uOldState:UINT
19     DEF uChanged:UINT
20     DEF ptActionx:INT
21     DEF ptActiony:INT
22     DEF lParam:INT
23 ENDTYPE
24 TYPE LVITEM
25     UINT mask
26     INT iItem
27     INT iSubItem
28     UINT state
29     UINT stateMask
30     POINIER pszText
31     INT cchTextMax
32     INT iImage
33     INT lParam
34     INT iIndent
35     INT iGroupId
36     UINT cColumns
37     INT puColumns
38 ENDTYPE
39 CONST LVM_FIRST = 0x1000
40 CONST LVM_GETTEXT = 0x1001

```

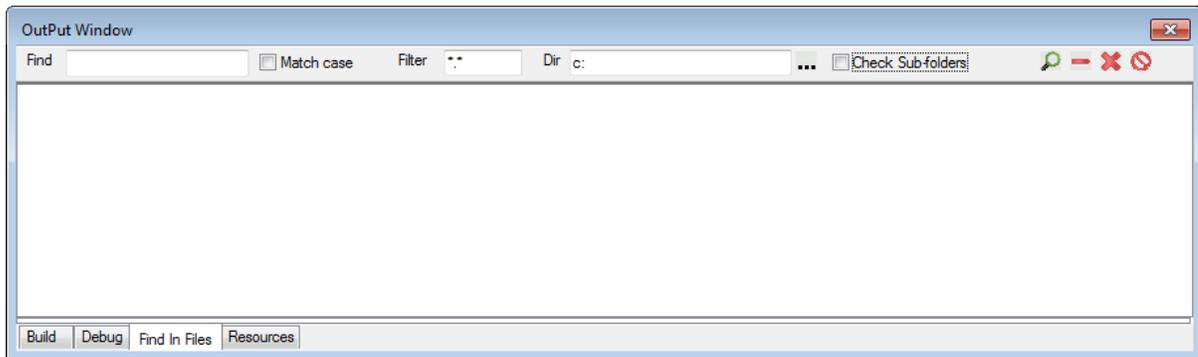
Zoom Level 2 - 90% of default dialog size



```
Print Code - C:\!IDE3\Cleaner.iwb
1  /*'=====
2      IW BASIC 3.X IDE
3      CLEANER FILE
4      VER 0.01 BETA
5      BY LARRY MCCAUGHN
6      COPYRIGHT 2012, IONIC WIND SOFTWARE  ALL RIGHTS RESERVED
7  '=====*/
8  $INCLUDE "stdafx.inc"
9  DIALOG addext
10 SETID "BN_CLICKED",0
11 TYPE NMLISTVIEW
12     DEF hwndFrom:UINT
13     DEF idFrom:INT
14     DEF code:INT
15     DEF iItem:INT
16     DEF iSubItem:INT
17     DEF uNewState:UINT
18     DEF uOldState:UINT
19     DEF uChanged:UINT
20     DEF ptActionx:INT
21     DEF ptActiony:INT
22     DEF lParam:INT
23 ENDTYPE
24 TYPE LVITEM
25     UINT mask
```

Zoom Level 3 - Text at full size

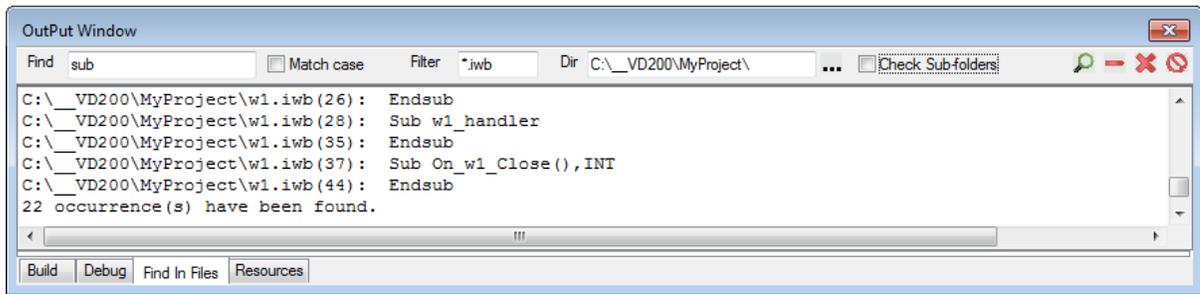
The following is an example of the *Preview Window* with the *Color* option selected.



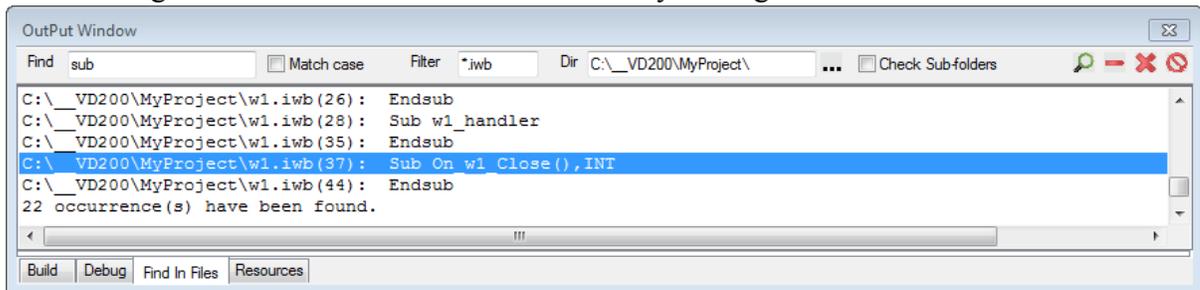
Each element is described below:

Item	Description
Find What	Text field containing the Search Term.
Filter	Text field containing the filter identifying what types of files to search in.
In Folder	The full path to the folder to search in.
...	Clicking the button will open a standard <i>Folder Browse</i> dialog. The folder selected by the User will appear in the <i>In Folder</i> text field when the dialog is closed.
Match Case	If checked, the search will look for exact case matches to the <i>Search Term</i> as it is entered by the User.
Look in Sub-Folders	If checked, the search will proceed through all nested sub-folders of the folder shown in the <i>In Folder</i> text field.
	Initiates the actual search. As matches are found entries are made in the <i>Results</i> listbox.
	The User can select one or more lines in the <i>Results</i> listbox (that are no longer needed) for deletion. Clicking this button will delete all lines that have been selected.
	Clicking this button clears ALL lines in the <i>Results</i> listbox.
	Used to cancel an in-progress search. Button does not appear until a search is started and disappears as soon as a search has ended.

The following shows a typical search. Notice that each match is indicated by filename, line number, and the text for that line.



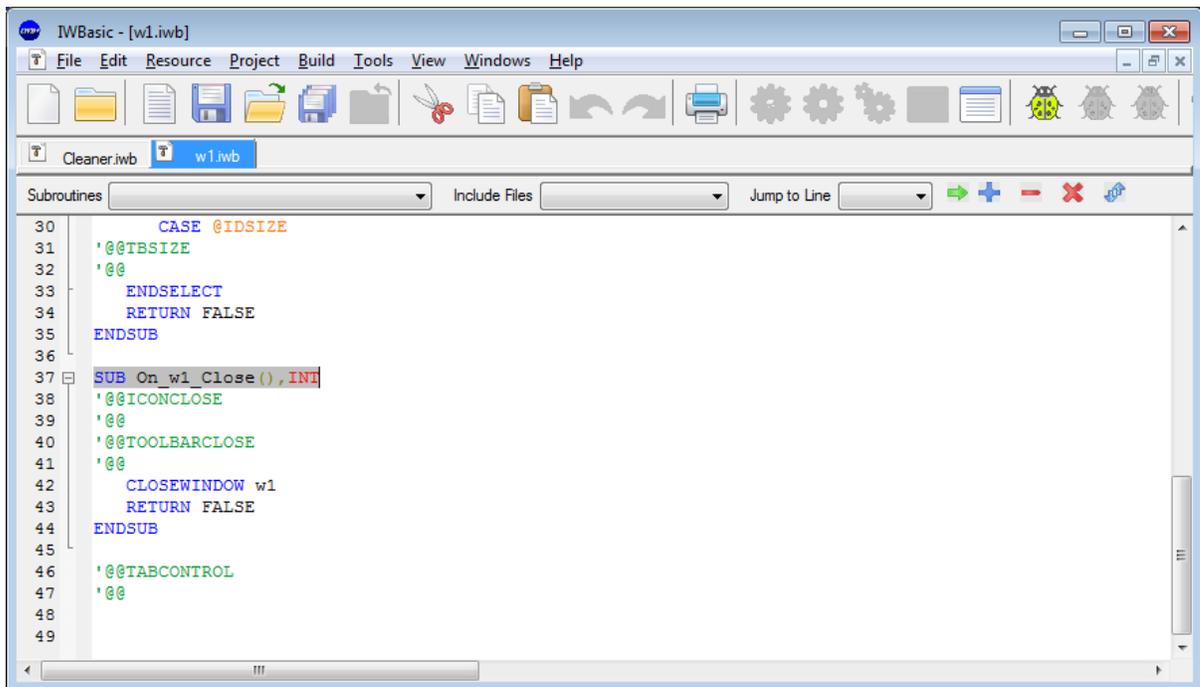
The following shows that the User has selected a line by clicking on it.



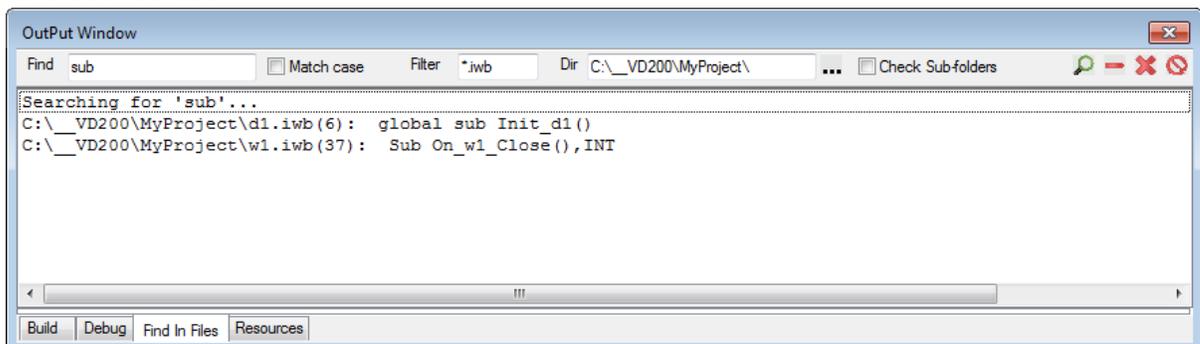
Double-clicking the selected line will result in the following:

- If the source file identified in the match line is not currently open it is opened in a new [Code Editor](#) window and made the active window.
- If the indicated file is already open it is made the active window.
- The [Code Editor](#) window is scrolled, as necessary, to insure the indicated line is visible.
- The indicated line is highlighted.

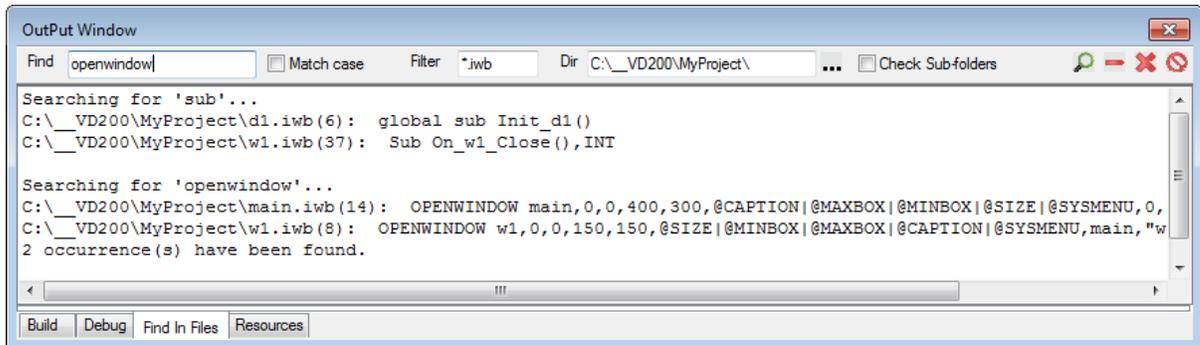
The following shows the resulting [Code Editor](#) window after the above steps.



When specific match lines are no longer needed they can be selected (one or more at a time) and deleted by clicking the  button. The following shows the above search results after some lines have been deleted.



The *Find-in-File* utility allows Users to execute unlimited, multiple searches without having to clear the previous search's results first. The following shows the results of an additional search without clearing the results above.



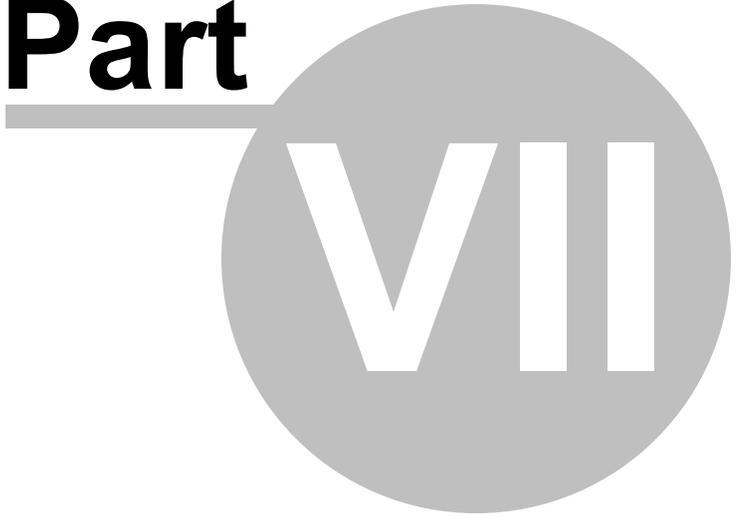
NOTE: There are three ways that all search results can be cleared.

1. Select all the lines in the *Results* listbox and click the  button
2. Click the  button without having to make any selections.
3. Closing and restarting the IDE.

Warning: Searching for very common words and/or from the root directory may, in very rare cases, cause the IDE to crash due to lack of memory.

To-Do Lists

Part



7 To-Do Lists

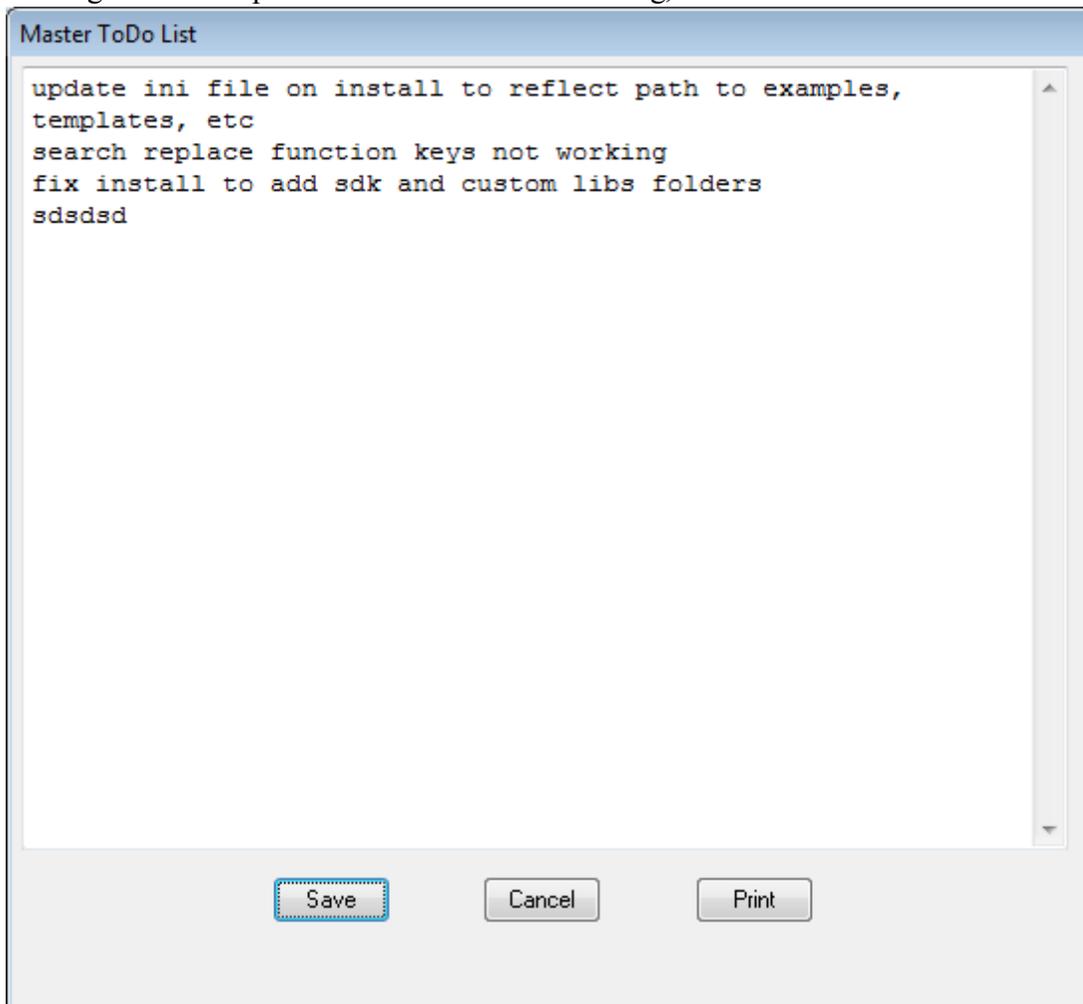
A *To-Do List* is exactly what the name implies; a list of tasks that need to be accomplished. For the convenience of the User, IWBASIC supports two types of *To-Do List*:

- [Master ToDo List](#)
- [Project ToDo List](#)

Master ToDo List

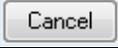
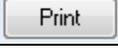
The lone *Master ToDo List* is available to the User at all times by clicking the  button on the [Main Toolbar](#).

Clicking the button opens the *Master ToDo List* dialog, shown below.



In addition to the area for entering tasks, the dialog contains three buttons discussed below.

Button	Description
--------	-------------

	Saves the dialog text to a file named ' <i>master_todo.txt</i> ' and then closes the dialog.
	Discards any changes after confirmation and then closes the dialog.
	Prints the text contents of the dialog.

Project ToDo List

Each open project has a unique *Project ToDo List* available to the User by selecting *Project / Project ToDo List* in the [Main Menu](#).

Selecting the menu option opens the project specific *Project ToDo List* dialog. This dialog functions exactly like the *Master ToDo List* shown above.

When the text is saved it is to a file name consisting of the *project name* + "_todo.txt" located in the same folder as the project file.

Language

Part



8 Language

8.1 Language Syntax Overview

The IWBASIC compiler supports an advanced version of the original BASIC language. Throughout this help document we will use the term IWBASIC to refer to the IWBASIC language and development environment.

Source Files

An IWBASIC source program is a collection of commands, statements, functions and preprocessor directives. The source may be contained in one file or separated among many. Each line in a source file can consist of one or more of the listed elements, a comment, or blank space. Multiple source lines may appear on the same editor line by separating them with a colon '!'. The end of a line of source code is always terminated by a return character.

Source lines may be continued into the next editor line by appending an underscore `_` after a comma or white space at the end of a line. Underscores are also allowed to be part of a name or identifier.

Source files need not contain executable statements. For example, you may find it useful to place definitions of variables in one source file and then declare references to these variables in other source files that use them. This technique makes the definitions easy to find and update when necessary. For the same reason, constants, DLL declares, COM interface definitions, and TYPE statements are often organized into separate files called "include files" or "header files" that can be referenced in source files as required.

Identifiers

An identifier is a term used to describe a token, variable name, function name, command, or statement. Identifiers can start with an underscore symbol or a letter. Identifiers can be an unlimited number of characters in length. An identifier cannot begin with a numeric but can contain any number of alphanumeric symbols after the initial symbol.

Example identifiers:

MyVariable
_Some_Function
LovePotion9

Reserved Words

The built in commands, statements and functions that make up the standard IWBASIC command

set are considered reserved words and cannot be used as an identifier in your own program. Reserved words are syntax highlighted in the editor to differentiate them from other language elements. IW BASIC supports installable command sets that extend the capabilities of the language. The installable command set becomes new reserved words for the language and will be appropriately highlighted in the editor.

Commands, Statements and Functions

In this document we will commonly refer to commands, statements and functions. In reality all of them refer to a subroutine in either a command set, external library, DLL, or COM object. IW BASIC treats command sets slightly differently than user subroutines or DLL functions.

A *command* is a subroutine with no parameters and can be used simply by using the name.

A *statement* is a subroutine with parameters that can generally be used without enclosing the parameters in parenthesis. Statements do not usually return values. Parenthesis are required around a statements parameters if the first parameter of that statement begins with a parenthesis of its own. The exception to this rule is the PRINT statement which neither requires or allows parenthesis surrounding the parameter list.

A *function* is a subroutine with parameters that returns a value. The parameters must be surrounded by parenthesis.

Examples:

```
'command
END
'statement
PRINT "Hello"
'function
A$ = STR$(99)
'a statement requiring surrounding parenthesis
LOCATE( (y+5) , x )
```

Parameters for statements or functions are separated by commas.

```
PRINT "HELLO ", "THERE"
```

A function in a command set is allowed to be used without parenthesis if the return value is ignored, that is to say not assigned to a variable.

```
OPENFILE myfile, "C:\\temp.txt", "W"
error = OPENFILE(myfile, "C:\\temp.txt", "W")
```

Resolving syntax conflicts

For compatibility with other languages parameter-less functions are allowed to be used without parenthesis for simple assignment operations and as an argument to another function if used by

itself. For example TIMER is actually a function and the correct syntax for using it would be:

```
i = TIMER( )  
PRINT TIMER( )
```

However the compiler will allow:

```
i = TIMER  
PRINT TIMER
```

When a parameter less function is itself used as an argument to another function or command the compiler may generate a "wrong number of parameters" error. This commonly occurs if a subtraction operation is performed on the value returned from a parameter less function.

```
' this produces an error  
PRINT TIMER-1
```

The error can be resolved in two ways. Use an empty parenthesis set for the function or surround the parameter less function with parenthesis. The former is the preferred method

```
PRINT TIMER( ) - 1  
PRINT (TIMER) - 1
```

Source comments

Comments will show up as green text in the editor and are ignored by IWBASIC. Comments are great for documenting your code so you can more easily understand what you wrote when you come back to it after a long period of time. The IWBASIC compiler supports single line and block comments

Single line comments begin with either a single quote mark or the REM statement

```
REM this is a comment and will be ignored  
'this is also a comment
```

Block comments, sometimes referred to as 'C' style comments begin with a /* character sequence and end with the */ character sequence. Block comments may extend across as many lines as necessary

```
/* this is a very long  
comment. all of this will be  
ignored by the compiler and is  
very convenient when we have  
complex code to document */
```

8.2 Constants and Literals

Constants are identifier to numeric mappings that cannot change value. Similar to variables in that

they are referred to by name but don't actually use memory or generate any code. The compiler substitutes a constant identifier with its numeric identity or string literal at compile time. The three built in statements for defining constants are [CONST](#), [SETID](#), and [\\$DEFINE \#DEFINE](#).

The `CONST` statement creates traditional name to value mappings common in other languages. `SETID` is unique to the IW BASIC language to create built in constants for the language accessed with the '@' operator. `SETID` constants are used throughout the command sets for built in function constants and are specially highlighted in the IDE. `SETID` constants created by the user are not highlighted in the IDE.

The `$DEFINE \#DEFINE` operator is normally used to create a conditional identifier, but may also be used to create the same kind of constants that the `CONST` statement does. When using `$DEFINE \#DEFINE` to create a constant the equal sign is not necessary.

Examples:

```
CONST WM_USER = 0x400
SETID "RTEDIT", 0x200
CONST ID_GAME = "Blaster Version 1.25"
CONST PI = 3.1415
$DEFINE ID_MENU 100
PRINT WM_USER, @RTEDIT, ID_GAME, PI
```

`CONST` and `$DEFINE \#DEFINE` allow basic math operations to calculate a numeric identity. The following operators may be used: =, ==, <>, !=, +, -, *, /, %, |, OR, || (xor), &, &&, AND, <<, >>, !, ~, ^, >, >=, <, <=. Since the final result must resolve to a fixed numeric value you cannot use functions or variables when calculating constants. You can however use other constant identifiers in the calculations.

```
CONST WM_USER = 0x400
CONST MY_MESSAGE = WM_USER + 1
CONST MY_MESSAGE2 = MY_MESSAGE + 1
```

[ENUM](#) is another method of defining constants. `ENUM` allows the creation and modifying of long lists of constants in alphabetical order without having to retype constant values. The same math operators may be used as with `CONST` and `$DEFINE \#DEFINE`.

String literals

A string literal is text enclosed in quotes. The compiler supports string escape sequences to insert special ASCII values into the string without having to use `CHR$` to generate them. String literals are stored in the executable when compiled.

Examples:

```
Mystring = "This is a string"
' a string with embedded quotes
MyString2 = "This \"is\" a string"
```

All escape sequences begin with a single backslash '\'. Because of this in order to have a backslash in the string itself you need to use a double backslash '\\'. This is important to remember when working with filenames.

```
Myfile = "C:\\data\\inp.text"
```

Supported escape sequences:

Sequence	Inserted character
\n	New line character
\t	TAB character
\\	A single backslash
\"	A quote
\octal	An ASCII character whose octal value can be up to 377 (377 _o =255)
\xnn	An ASCII character whose hex value is 'nn'.

The last two sequences deserve a bit of explanation. In other BASIC's it was necessary to use CHR\$ to insert a non printable character into a string. For example the ESC character is 0x1B in hexadecimal and 33 in octal. A\$ = CHR\$(0x1B) . With IWBASIC you can insert the character directly into the string by specifying its code in octal or hex. This saves time and results in smaller programs.

Example: "\377" - results in one character string, same as "\xFF"

Example: "\400" - results in two characters "\40" and "0", because 400_o=256 (is greater than 255)

```
A$ = "\x1BThere is an <ESC> character at the beginning"
A$ = "\33There is an <ESC> character at the beginning"
```

The maximum length of a string literal is 1023 bytes. If you need a longer string literal then append them together with the concatenation operator (+). It is quite unusual to have a string literal this long, consider using DATA statements to separate your strings.

Unicode literals

A Unicode literal string is prefixed with the L character. This tells the compiler to convert the literal to UTF-16 and return a Unicode string.

```
UNAMES$ = L"John Smith"
```

Supported escape sequences:

Sequence	Inserted character
\n	New line character
\t	TAB character

\\	A single backslash
\"	A quote
\octal	An ASCII character whose octal value can be up to 177777 (177777o = 0xFFFF)
\xnnnn	An Unicode character whose hex value is 'nnnn'. A max of FFFF

"T" string literals

A "T" literal string is prefixed with the T character. This tells the compiler to convert the literal to a Unicode string or an ANSI string based upon whether or not UNICODE is defined..

```
NAME$ =T"John Smith"
#ifdef UNICODE
declare import, testme alias testmeW(wstring p)
#else
declare import, testme alias testmeA(string p)
#endif

testme(T"hello")
print NAME$
```

"T" literal strings may include the same escape sequences that the ansi and unicode strings use. However, extreme care should be taken when using the octal and hex sequences. See TSTRING, ITSTRING

Numeric literals

Numeric literals are sometimes called numeric constants. We use the term literals to differentiate them from the CONST keyword and to avoid confusion. A numeric literal is simply a number, either integer or floating point, entered directly into the source code. Such as when assigning to a variable:

```
A = 1.3452 : REM The number is the literal.
```

Direct entering of exponents is also allowed by the compiler

```
A = 1.2e-10: REM Set a to equal 0.00000000012
```

Hexadecimal numbers may also be directly specified by using the 0x identifier

```
A = 0x500 + 0x2FFF
```

The older form of &h is still available but only if the hexadecimal number starts with a numeric and not a hex letter

```
A = &h1FFF: REM Acceptable
A = &hFFFF: REM Not acceptable, use a leading 0 as in &h0FFFF
```

Numeric modifiers

The compiler supports modifiers to the entered number to change its *type*. If a number is entered without a decimal point it is treated as an INT type (4 bytes) and with a decimal point as a DOUBLE type (8 bytes). It is sometimes desirable to modify the type of the numeric to tell the compiler how to convert and store it. A standard INT literal is not large enough to hold a UINT64 literal for example. Modifiers appear as one, two, or three characters following the literal.

```
REM INT64, A will be defined as an INT64
REM if not already defined
A = 36674965736284q

REM FLOAT. flNum will be defined as type FLOAT
REM if not already defined
flNum = 1.234f
```

Supported modifiers:

Modifier	Result
(none) w/o decimal	INT
(none) w/ decimal	DOUBLE
u	UINT
q	INT64
uq	UINT64
f	FLOAT
b	INT
bu	UINT
bq	INT64
buq	UINT64

The *u* modifier was specifically designed for entering hexadecimal numbers as an unsigned quantity. Normally all non decimal numbers are treated as a signed integer which can cause calculation problems when using hexadecimal numbers.

```
A = 0xFFFFFFFFFu -1u
```

Without the *u* modifier, A, if not already defined, would have a type of INT and a value of -2. By specifying the *u* modifier A is defined as a UINT with a value of 0xFFFFFFFFE or 4294967294.

The same sort of thing applies to the other qualifiers:

```
autodef = 1234567890uq
if (typeof(autodef) = @TYPEINT64)
print "type of 1234567890uq is INT64"
elseif (typeof(autodef) = @TYPEUINT64)
print "type of 1234567890uq is UINT64"
```


Program global variables are the same as source global but are made usable to any source file in your program by using the [GLOBAL](#) or [PROJECTGLOBAL](#) keywords.

Local variables

Local variables are defined within a subroutine between the [SUB](#) and [ENDSUB](#) keywords and are only usable by the code within that subroutine. The [STATIC](#) keyword can be used to define static variables in a subroutine.

```
sub function()
POINT pt
int x
static def y as int
static dim z as int
....
return
endsub
```

Defining variables

Variables must be defined before use either with the [DEF / DIM](#) statement or automatically defined by assigning a value. Automatic definition of variables can be disabled using the [AUTODEFINE](#) statement. Variables can also be defined using the reverse type method by specify the variable type first, followed by the identifier.

Examples

```
DEF name as STRING
name = "John Doe"
'Defines an integer variable by automatic assignment definition.
myNumber = 1
'Reverse type method. Identical to DEF name2 as STRING
STRING name2
```

Program global variables located in other source files can be accessed using the [EXTERN](#) keyword in place of [DEF / DIM](#). Create a program global variable by using the [GLOBAL](#) keyword along with the [DEF / DIM](#) statement.

```
'in file1
GLOBAL _gCodeName
DEF _gCodeName as STRING
...
'in file 2
EXTERN _gCodeName as STRING
PRINT _gCodeName
```

Variable types

A variables *type* tells the compiler what kind of data a variable can contain. IWBASIC supports numerous built in types and user types can also be described using the [TYPE](#) statement. Once a

variable is defined to a specific type it cannot be changed to a different type. The exception of this is the POINTER type which can point to any data type using type-casting. Built in variable types are reserved words and color highlighted in the editor as red by default. See [Figure 1](#) for a list of built in variable types

Variable Assignments

Variables are assigned values either directly with the = symbol or indirectly by a function or statement. Variables can be assigned the contents of other variables as long as they are the same *type* or an appropriate conversion can be performed by the compiler. For example, an integer variable can be assigned the contents of a float variable but will lose its fractional value:

```
DEF WorkDays:FLOAT
DEF Days:INT
WorkDays = 13.5
Days = WorkDays
PRINT Days
```

Will print the number '13' as the fractional value of '.5' was lost in the conversion of FLOAT to INT.

Any assignment conversions that cannot be performed by the compiler will generate an *Invalid Assignment* error in the Build window of the IDE. An example of an invalid assignment would be trying to assign a number to a string

```
DEF Name$,Name2$:STRING
DEF Initial:CHAR
Name$ = "John Doe"
Name2$ ="Julie Doe"
Initial = "A"
Name$ = Name2$
REM this next line produces an error
Name2$ = 100
```

The compiler will also generate an error if you try and assign a value to a constant.

Post assignment

IWBASIC allows assigning variables as they are defined, which can be used to give a variable an initial value without a separate assignment statement.

Examples:

```
INT a = 100
STRING name = "Jerry"
DEF b=100.0 as DOUBLE
```

Global Initializers

The % symbol can be used to initialize a variable without it needing to be in the path of execution. When used the variable will be located in the initialized data segment of the executable. This also allows placing project global variables in separate source files without needing to execute any code.

Examples:

```
string %version="My Program 1.0"  
global uint %timeout=200
```

WARNING: If you use % to define a string variable then that string will only be as long as the initializer, consider strings initialized like this as read only.

NOTE: Using %variablename is evaluated at compile time, not run time, in as much initializers must be constant, or resolve to a constant.

Arrays

An array is a collection of data of the same type. Arrays are referenced by their variable name and an index. Think of a box of cookies. Each row of cookies can be thought of as an array.

IW BASIC supports arrays up to **five** array dimensions. When using an array the index is enclosed in brackets '[']'.

```
DEF age[20]:INT  
age[0] = 40  
age[1] = 32
```

As noted in the above example you access an array using an 'index' into the array. Indices start at 0 so the maximum index will be one less than the value specified in the DEF / DIM statement. To create a multidimensional array use the comma to separate dimensions.

```
DEF myarray[20,20]:INT  
DEF BigArray[10,10,10,10,10]:FLOAT  
myarray[0,1] = 5
```

An array may be initialized by using a list of data parameters. Each line can have up to 100 data elements and the array can be initialized starting with a certain element.

```
DEF myarray[10]:INT  
'initialize the first 7 elements  
myarray = 1,10,23,2,4,16,27  
'initialize the last 3 elements  
myarray[7] = 5,18,42
```

Array dimensions in the DEF / DIM statement must either be a numeric constant or resolve to a constant when compiling. Dynamic arrays are supported by using the [NEW](#) statement or using memory with [ALLOCMEM](#).

The ISTRING type

ISTRING is an advanced STRING type that can be defined and accessed like an array. ISTRING can then be used anywhere a normal string would be expected. Unlike a normal STRING variable the maximum length of an ISTRING is limited only by available memory.

```
OPENCONSOLE
'Define a string of exactly 30 characters
DEF name[31]:ISTRING
name = "john Smith"
name[0] = "J"
PRINT name
PRINT "Press any key to close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The IWSTRING type

IWSTRING is an advanced WSTRING type that can be defined and accessed like an array. IWSTRING can then be used anywhere a normal Unicode string would be expected. Unlike a normal WSTRING variable the maximum length of an IWSTRING is limited only by available memory.

```
OPENCONSOLE
'Define a string of exactly 30 characters
DEF name[30]:IWSTRING
name = L"john Smith"
name[0] = L"J"
PRINT W2S(name)
PRINT "Press any key to close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The TSTRING type

TSTRING is an advanced STRING type that is used to write unicode+ansi friendly programs. Use a TSTRING to generate a STRING or WSTRING, based on the current UNICODE condition. If the condition is defined, TSTRING will generate an unicode WSTRING, else an ansi STRING.

```
TSTRING = T"hello"
#ifdef UNICODE
' the first line resolves to WSTRING = L"hello"
#else
' the first line resolves to STRING = "hello"
#endif
```

The ITSTRING type

ITSTRING is an advanced ISTRING type that is used to write unicode+ansi friendly programs. Use a ITSTRING to generate an ISTRING or IWSTRING, based on the current UNICODE condition. If the condition is defined, ITSTRING will generate an unicode IWSTRING, else an ansi ISTRING.

```
OPENCONSOLE
$DEFINE UNICODE
'Define a string of exactly 30 characters
DEF name[30]:ITSTRING
name = T"john Smith"
name[0] = T"J"
PRINT name
PRINT "Press any key to close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The OEM type

OEM is an advanced STRING type that is used to create strings with diacritical characters, which can be displayed in console window. Internally the string is encoded to an unicode string using the current codepage, then the CharToOemW api is called, to convert it to console OEM. In Windows ME/98/95 this api is supported by the Microsoft layer for unicode which will need to be installed on the older systems (unicows.dll).

On Windows 2000 or newer, the required api is available in user32.dll

```
print OEM"?□□□□□"
```

User defined variable types (UDT)

The TYPE statement begins defining a user variable type. The new type will contain all of the variables between TYPE and ENDTYPE. Accessing the variables in a user type is done with the dot operator ‘.’. Any variable type can be defined between TYPE and ENDTYPE including nested TYPE/ENDTYPE definitions. One or more [UNION](#) and [ENDUNION](#) definitions may be used. Also, all the [\\$IF](#) directives are allowed inside TYPE-ENDTYPE definitions.

Once a user variable type is defined you can create variables of that type using the DEF / DIM statement or by assigning. IW BASIC UDT's conform to ANSI standards for element packing. The element packing value can be changed if needed, see the [TYPE](#) command for details.

```
TYPE phonerecord
  DEF Name:STRING
  DEF Age:INT
  DEF Phone[20]:ISTRING
ENDTYPE
DEF Rec:phonerecord
Rec.Name = "Joe Smith"
Rec.Age = 35
```

```
Rec.Phone = "555-555-1212"
...
```

All strings and arrays are stored by value in an IW BASIC UDT. If you require a reference to a string in a UDT for an API function use the POINTER type in place of the STRING type.

The following operators can be used to compare UDT's: =, ==, <>, !=, <, >, <=, >=

UDT's may be initialized with a single-byte initializer (instead of calling RtlZeroMemory):

```
STARTUPINFO si = 0 ' up to 255
STATDATA sdata = 0
```

Unions

A union is a special form of UDT where all member variables share the same memory. The size of a union is the size of the largest member aligned to the element packing value. IW BASIC supports both unions (named and un-named) embedded in TYPE definitions and named unions defined outside of a TYPE definition. Unions are used primarily as a memory saving technique when separate UDT definitions might differ by only the type of a single member variable. Examples:

```
TYPE number
  INT numtype
  UNION
    FLOAT flValue
    UINT uiValue
  ENDUNION
ENDTYPE

DEF n1,n2 as NUMBER
n1.numtype = 1: '1 = float, 2 = uint
n1.uiValue = 1002
n2.numtype = 2
n2.flValue = 123.45f
```

In the above example both flValue and uiValue share the same memory within the UDT. Only one is in use at any time. The size of the UDT is 8 bytes, instead of 12 if we did not use the union.

See Also: [UNION](#), [ENDUNION](#)

Creating aliases for variable types with TYPEDEF / \$TYPEDEF

The TYPEDEF / \$TYPEDEF (type define) statement allows referring to a built in variable type by a different name. This can aid in code readability and conversion of source code from other languages. For example some common variable types found in C are 'byte', 'bool', 'HANDLE', etc. which are themselves aliases for an unsigned character, integer and unsigned integer. These type names can be duplicated in IW BASIC using the following statements:

```
TYPEDEF byte CHAR
```

```

TYPEDEF bool INT
TYPEDEF HANDLE UINT

```

You can also use a previous type define to create a new one as in:

```

TYPEDEF HWND HANDLE

```

Also valid are:

```

TYPEDEF newtype structureName

```

```

TYPEDEF newtype className

```

```

TYPEDEF newtype interfaceName

```

```

TYPEDEF MYPOINT POINT

```

is the same as

```

type MYPOINT

```

```

    int x

```

```

    int y

```

```

endtype

```

Once a variable type has been aliased with TYPEDEF / \$TYPEDEF it can be used in place of the actual type.

```

DEF bDone as bool
DEF hWnd as HWND

```

It is important to note that the name does not change how the variable is used. Internally after the above is executed a bool type would still be an INT. Its just a convenient way of keeping track what a variable is being used for.

Figure 1. Built in variable types:

TYPE	AUTODE FINE	USAGE	DIRECT ASSIGN	SIZE IN BYTES
WORD	NO	Unsigned short integer from 0 to 65535	YES	2
SWORD	NO	Signed short integer from -32,768 to 32,767	YES	2
INT	YES	Signed integer -2147483648 to 2147483647	YES	4
UINT	YES	Unsigned integer 0 to 4294967295	YES	4
INT64	YES	Signed large integer -9223372036854775808 to 9223372036854775807	YES	8
UINT64	NO	Unsigned large integer 0 to 18446744073709551615	YES	8

FLOAT	YES	Single precision numbers	YES	4
DOUBLE	YES	Double precision numbers	YES	8
STRING	YES	Text up to 254 characters	YES	255
WSTRING	YES	Unicode text up to 254 characters	YES	512
OEM	NO	Diacritical text up to 254 characters	YES	255
TSTRING	YES	Resolves to either a STRING or WSTRING based upon definition of UNICODE	YES	255/512
ISTRING	NO	Dimensionable string.	YES	Variable
IWSTRING	NO	Dimensionable unicode string.	YES	Variable
ITSTRING	NO	Resolves to either a ISTRING or IWSTRING based upon definition of UNICODE	YES	Variable
CHAR	NO	Single character of text or any number from 0 to 255	YES	1
SCHAR	NO	Signed character. Integer value from -128 to 127	YES	1
WINDOW	NO	WINDOW UDT	NO	N/A
DIALOG	NO	DIALOG UDT	NO	N/A
FILE	NO	ASCII file I/O	NO	N/A
BFILE	NO	Binary file I/O	NO	N/A
MEMORY	NO	Allocated memory	YES	N/A
POINTER	NO	Variable reference	YES	N/A
COMREF	NO	COM object pointer	YES	4
DATABLOCK	NO	Reserved. Used by DATA functions.	NO	N/A
ANYTYPE	NO	Variable subroutine parameters.	NO	8
HEAP	NO	Reserved. Special return pointer used by string functions.	YES	4
WHEAP	NO	Reserved. Special return pointer used by string functions.	YES	4

8.4 Operators

An operator is a symbol that performs a specific function on a variable, constant or identifier. Operators consist of mathematic operators, conditional operators, Boolean operators and control operators. Some symbols are reused for other purposes depending on the context of the operator. For example the & symbol is used as the *bit wise AND* operator and also as the *Address Of* operator

Mathematic operators

The IW BASIC compiler understands the following table of mathematic operators.

Operator	Function
+	Addition
-	Subtraction, unary minus
*	Multiplication
/	Division
&	Bit wise AND
&&	Bit wise AND
	Bit wise OR
	Exclusive OR (XOR)
%	Modulus. The remainder of an integer division
^	Raise to a power
<<	Bit shift left
>>	Bit shift right
!	Logical NOT (Boolean Negation)
~	Bit wise NOT - supported in constants and variables up to 32-bit: (s)char,(s)word,(u)int

```
SetWindowLong(hwnd, GWL_STYLE, styles & ~WS_VISIBLE) ' hide window
```

Every mathematic operator requires a left hand and right hand operand with the exception of the unary minus operator which expects only a right hand operand. Operations may be ordered with parenthesis.

Compound operators and prefix/postfix operators

Compound operators, also called assignment operators, use one mathematic operator in conjunction with an assignment to create a shortcut to common expressions. For example it is very common in programming to add a number to a variable and assign that number to itself as in $A=A+5$. The compound operators shorten this by only requiring the variable name once, so to add 5 to a variable you can specify $A += 5$.

The following table lists the compound operators:

Operator	Result
+=	Add an expression
-=	Subtract an expression
*=	Multiply an expression
/=	Divide by an expression
&=	AND with an expression
=	OR with an expression

++	Increment variable by 1
--	Decrement variable by 1

Examples:

```
A++ : REMAdd one to A
MyVar-- : REM Subtract one from MyVar
A /= 2 : REM divide A by 2
C *= 3: REM Multiply C by 3
if (a == b) then equal()
if (a != b) then notEqual()
if (a && b) then BothTrue()
if (!a && !b) then BothFalse()
```

The increment (++) and decrement (--) operators can be applied as both prefixes and postfixes.

```
variable++
variable--
++variable
--variable
```

All the expressions return a value, which can be used directly in another expression:

Print numbers 3,2,1,0

```
int v = 4
while (v-- > 0)
    print v
endwhile
```

The difference between prefix and postfix:

```
int x = 4
print x++ ' will print 4, and x will be set to 5. (Return current value, then modify)
int x = 4
print ++x ' will print 5, and x will be set to 5; (First modify, then return new value)
```

Note: when used as function arguments, the execution flows from right to left :

```
int x = 4
print x++, x++
print x
```

will result in:

```
5, 4
6
```

```
int x = 4
print ++x, ++x
print x
```

will result in:

```
6, 5
6
```

Conditional operators

Conditional operators are used to compare variables or constants and return either TRUE (1) or FALSE (0) as a result. Conditional operators are normally used by conditional statements but also can be used in conjunction with mathematic operators. The following table lists the conditional

operators understood by the compiler

Operators	Meaning
>	Greater Than
<	Less Than
<>	Not Equal To
=	Equal To, Assign To (See note)
==	Equal To, Assign To (See note)
!=	Not Equal
>=	Greater Than or Equal To
<=	Less Than or Equal To

Note: In [macros](#), "=" is used to assign and "==" is used to test for equality.

Example usages:

```
IF A < 5 THEN PRINT "A < 5"
if (a = b) then equal()
DO:UNTIL INKEY$ <> ""
```

Boolean operators

IWBASIC supports two Boolean operators. *AND* and *OR*. The operators return either TRUE or FALSE and are generally used by conditional statements to group two or more conditional tests together. AND and OR are blocking operators which means if the first conditional test is FALSE for AND, TRUE for OR, then the second conditional test will not be executed.

Examples of blocking:

```
A=0 : Z=1
'First case A <> 1 so the second test Z=0 wont be evaluated, result is FALSE
IF (A=1) AND (Z=0) THEN PRINT "This won't be printed"
'Second case A=0 so the second test Z=0 wont be evaluated, result is TRUE
IF (A=0) OR (Z=0) THEN PRINT "This will be printed"
```

Blocking is useful when performing the second test would be undesirable if the first test fails. A common usage is to test a pointer for NULL before de-referencing it.

Control operators

Control operators either cause code to be executed or return information about a function or identifier. The following control operators are supported by the IW BASIC compiler:

Operator	Description
&	Returns the address of a subroutine, string or wstring as a UINT
!	Indirectly calls a subroutine
->	Calls a COM method
#	De-references a pointer
*	C style pointer dereferencing
##	Advanced de-reference operator. Used with the ANYTYPE parameter or to use a UINT variable as a memory pointer.

Control operators will be described in detail elsewhere in the help document.

Operator usage with STRING variables

IWBASIC allows conditional comparison of strings. When comparing strings the allowed operators are:

Operator	Description
=	Test for equality
==	Test for equality
<>	Test for inequality
!=	Test for inequality
<	Test if one string is less than another alphabetically
>	Test if one string is greater than another alphabetically
<=	Test for less than or equal to
>=	Test for greater than or equal to

The less than and greater than variants check each character position until a determination can be made. The first character always has greatest significance in the test.

The *Address Of* operator, &, for strings and wstrings is usefull when passing a string/wstring as LPARAM in SendMessageA/W.

```
int address = &"text"
address = &L"text"
SendMessageA(dl.hwnd, WM_SETTEXT, 0, &"caption")
```

String evaluation

Strings up to 4 bytes (4 or less characters) in length can be evaluated. `A` will return 65, the same as ASC("A") and 0x41.

```
int i = `ABCD`
'i is equal to 1094861636 or 0x41424344 hex
istring s[4]
s[0] = `A`,0
```

Unicode strings up to 4 bytes (2 or less characters) in length can also be evaluated. L'A' is equal to 65.

```
iwstring w[4]
w[0] = L`Ã¼`
w[1] = L`Ã³`
w[2] = L`武`
w[3] = L` ` ' =0; =` ` ; =L` `
```

String concatenation operator

The maximum length of a string literal is 1023 bytes. The + operator can be used with literal strings and/or string variables to combine, or append them, together. The result of the operation is the combination of all the strings in the expression.

```
A$ = "This " + "is " + "a " + "String!"
PRINT A$
```

Also works with unicode strings

```
wstring A$ = L"This " + L"is " + L"a " + L"String!"
PRINT A$
```

See Also: [APPENDS](#)

String multiplication operator

The string multiplication operator is used in the following format : string*boolean

It will return a designated string if boolean is true, and an empty or NULL string if boolean is false

```
int count = 1
print "found ", count, " item", "s"*(count<>1)
rem result is 'found 1 item'
count = 2
print "found ", count, " item", "s"*(count<>1)
rem result is 'found 2 items'
```

Also works with unicode strings

```
int count = 1
print L"found ", count, L" item", L"s"*(count<>1)
rem result is 'found 1 item'
count = 2
print L"found ", count, L" item", L"s"*(count<>1)
rem result is 'found 2 items'
```

Operator Precedence

Operator precedence defines the order of execution of operators in an expression. The precedence can be overridden by using parenthesis to order the execution. Precedence at the same level is from left to right.

From highest precedence to lowest:

```
[ ]
*pointer, #pointer, *<type>pointer, #<type>pointer
- Unary Minus, !, ~
++, --
( )
^, #, ##
/, *, %
+, -
<< >>
<, >, <=, >=
=, ==, <>, !=
&
||, XOR
|
```

```
&&, AND
OR
?
:
```

Examples:

```
A = 1 + 2 * 4
```

Performs the multiplication first, followed by the addition with a result of 9

```
A = (1+2) * 4
```

Performs the addition first followed by the multiplication with a result of 12

```
A = 1 + 2 * 4 - 5
```

Multiplication is first followed by the addition and then the subtraction with a result of 4

Type Promotion

For all mathematical operations the resultant type will be the same as the operand with the highest precision. Operands can be given a different type for use in the calculation with the [INT](#) or [FLT](#) functions.

For example a common user programming error is dividing two integers and assuming the result will be a floating point value:

```
DEF b,c as INT
b = 1:c = 2
A = b / c
```

Will result in a value of 0 because both operands are integer. However if one of the operands is a floating point type the result will be 0.5. This can be accomplished by using either a type modifier or the FLT function.

```
A = b / FLT(c)
```

In this case c is temporarily promoted to a floating point value and the result will then be a floating point value.

8.5 Pointers and Typecasting

A pointer is a special type of variable that can reference, or point to, another variable. The pointer when initialized is said to *reference* another variable, or to contain a memory location. To understand pointers fully you need to remember some key concepts.

- 1) *A variable is really a memory address,*
- 2) *The memory address is where the data is stored.*

So when we assign a variable a value there are really two values involved. The first is the value we want to store, and the second is the starting address of the memory we want to store it in. That address is a value in and of itself. On 32 bit processors you can think of a memory address as a UINT type.

```
A = 1
```

Seems simple enough. Internally the compiler tells the linker that we want to move the number 1 into the address that A represents. That address is picked by the linker when your program is compiled and can be any where in your programs *address space*.

A pointer then is a variable that has an address, and in that address we store another address.

```
A = 1 : 'Store the number 1 in the address that A represents
DEF p as POINTER
p = A : 'Store the address of A into p
```

Retrieving the value that is contained in the address, that is stored in the pointer is called *dereferencing*. In some texts you may see it referred to as *indirection*. The IW BASIC compiler supports two general dereferencing operators, the # symbol and a 'C' style dereference operator, the '*'. The hash dereference '#' operator is unique to the IW BASIC language and is suitable for most basic pointer needs.

```
PRINT #<INT>p
```

The above statement may look a bit strange. Lets break it down by what it does.

- We want to perform a dereferencing operation

p - The pointer that contains the address of a variable

<INT> - The value stored in the address of the variable is of type INT

When we combine the two examples the dereferencing operation returns the number 1. The type between the < > is called *type casting*. The compiler must know what *type* is stored in the address being dereferenced. If the pointer was assigned directly as above you can omit the type cast and simplify the statement to:

```
PRINT #p
```

This only works if the pointer was assigned an actual variable and you haven't crossed a subroutine boundary with the pointer. The compiler will generate an error message of "Type cast must be specified" if it cannot automatically determine the type of the value an address contains. For pointers to a UDT variable (user data type) a typecast must always be specified.

You can determine the type that a pointer is currently referencing by using the [TYPEOF](#) function. If a pointer has never been assigned an address then TYPEOF will return -1.

Dereferencing operations can appear on both the left hand and right hand side of an assignment

operation. In this way you can indirectly change the contents of a variable by using the pointer. Continuing with the statements from above:

```
#<INT>p = 2
```

Stores the value of 2 into the address of the variable pointed to by p and that variable is of type INT.

It is important to note that if a pointer is a *NULL pointer* then any dereferencing operation will result in an *access violation* crash. A NULL pointer is a pointer that hasn't been initialized, or assigned an address yet. ***This is the most common reason that programs fail.*** If your unsure what a pointer will contain, such as a pointer passed to a subroutine, then always test your pointers for NULL before performing a dereference operation.

```
IF p <> NULL
    #<INT>p = 2
ENDIF
```

Type casting

Type casting was briefly mentioned in the last section. It is important enough of a topic to cover in further detail because the real power of pointers does not really become apparent until you realize that a memory address can be dereferenced as any built in or user defined type.

As mentioned above a type cast is the variable type between the <> symbols. A common use for type casting is using a block of memory to store any kind of data, structured or not.

```
DEF pMem as POINTER
pMem = NEW(CHAR,1000) : 'Get 1000 bytes to play with
#<STRING>pMem = "Copy a string into memory"
pMem += 100
#<UINT>pMem = 34234: 'Use bytes 100-103 to store a UINT
DELETE pMem
```

When accessing a UDT with a pointer you use the same dot notation you would with a statically defined UDT variable.

```
DEF pMem as POINTER
pMem = NEW(WINRECT, 1): 'WINRECT is built in
#<WINRECT>pMem.left = 0
#<WINRECT>pMem.top = 0
#<WINRECT>pMem.right = 100
#<WINRECT>pMem.bottom = 400
DELETE pMem
```

The above seems a bit cumbersome. Luckily you can use the [SETTYPE](#) command to preset the type of a pointer, so you only need to specify the type once:

```
DEF pMem as POINTER
pMem = NEW(WINRECT, 1): 'WINRECT is built in
```

```
SETTYPE pMem, WINRECT
#pMem.left = 0
#pMem.top = 0
#pMem.right = 100
#pMem.bottom = 400
DELETE pMem
```

Type casting can also be applied to all the basic numeric types via the following functions: `double()`, `float()`, `int64()`, `uint64()`, `int()`, `uint()`, `word()`, `sword()`, `char()`, `schar()`, `byte()`, `file()`, `bfile()`, `memory()`, `comref()`.

```
print double(1) ' prints 1.00
print byte(65+256) ' prints 65
```

Pointer math and array indexes

Since a pointer contains an address, and that address is a 32 bit value, you can change the address the pointer contains by using any standard math operator.

```
pMem = NEW (CHAR, 100)
FOR x = 1 to 100
  #<CHAR>pMem = x
  pMem++
NEXT x
```

The C style pointer dereferencing supports direct pointer math:

```
pMem = NEW (CHAR, 100)
FOR x = 0 to 99
  *<CHAR>(pMem+x) = x+1
NEXT x
```

It is important to remember that pointer math in this manner always refers to a single byte, regardless of the type cast. Multiply the size of the variable for more complex operations:

```
pMem = NEW (UINT, 100)
FOR x = 0 to 99
  *<UINT>(pMem+(x * 4)) = x+1
NEXT x
```

A simpler method to access a pointer to an array of data is to use the standard array indexes. In this manner you do not have to account for the size of the variable, as the type cast will specify the size of the data being stored.

```
pMem = NEW (UINT, 100)
FOR x = 0 to 99
  #<UINT>pMem[x] = x+1
NEXT x
```

Multiple indirection

Multiple indirection, or multiple dereferencing, is performing a dereferencing operation on a pointer more than once. In other words a pointer can contain the address of another pointer. IW BASIC supports multiple indirection through both the IW BASIC and C style operators. It is a bit easier with the C style. There really is no simple example of multiple indirection, however we will cover the basics.

To get an address of a pointer use the & operator:

```
DEF p1,p2,temp as POINTER
DEF num as INT
p1 = num: 'p1 contains the address of num
p2 = &p1: 'p2 contains the address of p1
'use multiple indirection to store the value of num
*<INT>*<POINTER>p2 = 5
'To use the IW BASIC style dereference use a temporary pointer
temp = #<POINTER>p2
#<INT>temp = 7
```

In this example the & operator gets the *address* of a variable. This is necessary when using pointers as an assignment of one pointer to another would normally copy the address contained by the pointer, not the pointers address itself.

The statement `*<INT>*<POINTER>p2 = 5` when broken down means "p2 is a pointer, it contains the address of a POINTER that contains the address of an INT, store the number 5 in that address".

There is no limit on the amount of indirection, although it would be unusual to see more than two levels of indirection. Text editors routinely use two levels of indirection to store lines of text. The first level is an array of pointers, and the second levels are pointers to strings containing each line of text. In this manner the text on each line can be adjusted individually without affecting any other line.

8.6 Conditional Statements

IW BASIC has two main conditional statements, IF and SELECT. A conditional statement selects a section of your program to execute based on a condition that you choose. There is also a special in-line version of the Windows IIF statement.

IF Statement

The IF statement will probably be your most used conditional statement. The statement has two forms, the block IF and a single line IF. The block form executes one or more statements if a condition is TRUE

```
IF condition {THEN}
'Statement(s) to execute if condition is TRUE
ELSE
```

```
'Statement(s) to execute if condition is FALSE
ENDIF
```

The condition is any math, comparison or algebraic expression that results in a yes or no result. The ELSE keyword is optional and represents the group of statements that will be executed if the condition is false. The THEN keyword in a block IF is also optional and is supported for backwards compatibility with other languages. Every block style IF must be matched with a corresponding ENDIF statement

An example of the IF block statement:

```
OPENCONSOLE
DEF number:INT
INPUT "Pick a number ",number
IF number = 12
    PRINT "You guessed correctly!"
    PRINT "Your very smart"
ELSE
    PRINT "Sorry wrong number"
ENDIF
PRINT "Press Any Key To Close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

In the previous [Operators](#) page we listed the conditional operators that can be used in an IF statement. Comparisons can be performed on numbers, strings and variables.

The IF statement also supports a single line version using the THEN keyword. When used on a single line the ENDIF statement is not necessary. Single line IF statements allow only one action be executed when the condition is true and one following an optional ELSE.

Example of a single line IF statement:

```
OPENCONSOLE
DEF number:INT
INPUT "Pick a number ",number
IF number = 12 THEN PRINT "You guessed correctly!" ELSE PRINT "Sorry wrong number"
PRINT "Press Any Key To Close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

You can use multiple conditions in an IF statement by using the Boolean operators AND and OR. The conditions should be enclosed in parenthesis because the Boolean operators have higher precedence. Examples:

```
IF (a < 1) AND (b > 3)
    PRINT "TRUE!"
ENDIF
```

Will print TRUE! if a is less than 1 AND b is greater than 3.

```
IF (a < 1) OR (b > 3)
  PRINT "TRUE!"
ENDIF
```

Will print TRUE! if a is less than 1 OR b is greater than 3.

ELSEIF statement

The ELSEIF clause allows combining multiple IF/ELSE/ENDIF statements into a single block. The condition of each ELSEIF statement is only compared if the previous IF or ELSEIF statement is FALSE. For example:

```
IF name = "Jerry"
  pay = "7.55"
ELSEIF name = "Tom"
  pay = "9.55"
ELSEIF name = "Lisa"
  pay = "10.34"
ENDIF
```

The above code is equivalent to:

```
IF name = "Jerry"
  pay = "7.55"
ELSE
  IF name = "Tom"
    pay = "9.55"
  ELSE
    IF name = "Lisa"
      pay = "10.34"
    ENDIF
  ENDIF
ENDIF
```

As you can see the ELSEIF statement saves a lot of typing.

SELECT Statement

The SELECT statement is an advanced conditional statement that allows you to test against many different combinations. The syntax of the select statement is:

```
SELECT expression
  CASE test1
    'Statement(s) to execute if expression = test1 is TRUE
  CASE test2
    'Statement(s) to execute if expression = test2 is TRUE
  CASE testn
    'Statement(s) to execute if expression = testn is TRUE
  DEFAULT
    'Statement(s) to execute if all tests are FALSE
ENDSELECT
```

The SELECT statement allows unlimited 'if equal to' conditions. The statements after the CASE

statement will only be executed if the expression is equal to the test condition. Example:

```

OPENCONSOLE
DEF Choice$:STRING
PRINT "Press some keys, Q to quit"
LABEL again
DO
Choice$ = INKEY$
UNTIL Choice$ <> ""
SELECT Choice$
  CASE "A"
  CASE& "a"
      PRINT "You pressed A!!"
  CASE "Z"
  CASE& "z"
      PRINT "Z is my favorite letter!"
  CASE "Q"
  CASE& "q"
      CLOSECONSOLE
      END
  DEFAULT
      PRINT "You pressed the letter ",Choice$
ENDSELECT
GOTO again

```

The DEFAULT condition is optional and will be executed if none of the CASE statements is true. CASE statements are mutually exclusive. To group conditional tests for the same statements together use the CASE& statement following the initial test CASE.

The SELECT statement can also be used to test multiple conditions and execute the first TRUE case. To do this use SELECT 1 as the opening statement and code each CASE statement as a condition. Example:

```

OPENCONSOLE
DEF Choice:INT
PRINT
LABEL again
INPUT "Enter a Number, 0 to end: ", Choice
IF Choice = 0
  CLOSECONSOLE
  END
ENDIF
SELECT TRUE
  CASE (Choice > 10)
      PRINT "number is greater than 10"
  CASE (Choice < 10)
      PRINT "Number is less than 10"
  CASE (Choice = 10)
      PRINT "Number is equal to 10"
ENDSELECT
PRINT
GOTO again

```

In this example we test the choice against multiple conditions using only one SELECT statement.

Your conditions can be as complex as needed, only the first TRUE condition will be executed so make sure the conditions are unique.

In-Line IIF Statement

The in-line IIF clause can be used to test any condition and choose one value or another based on whether the test is true or false. The syntax is as follows:

A ? B : C

Where:

A is the variable or expression being tested for a TRUE or FALSE state.

B is executed/returned if A is TRUE

C is executed/returned if A is FALSE

```
int myflag = true
print myflag ? "true" : "false"

int totalsales = 1000
int salesbudget = 1900
print (totalsales > salesbudget) ?_
"Exceeded budget by $" + str$(totalsales - salesbudget) :_
"Missed budget by ($" + str$(salesbudget - totalsales) + ")"
```

Note - the colon separates the TRUE and FALSE results and is not used as a separator for a new instruction.

B and C must be of matching TYPES. Mixing INT and FLOAT, FLOAT+DOUBLE, or any other "bad" combination is not allowed. The following will generate an error:

```
print boolean ? 1 : abs(5)
```

If B or C is a non-function and the other is a function, the function must return a value that matches the type of the non-function. If both are functions their returned types must match. Both functions having no return value is considered a match.

8.7 Loop Statements

IWBASIC has many built in loop statements. Loop statements execute one or more lines of your program repeatedly until a condition is satisfied, or until all elements have been iterated.

FOR Statement

The FOR statement executes lines of code until a counter variable reaches a specified number. The NEXT statement determines the end of the loop. The FOR statement has the syntax of:

```
FOR variable = start TO end {STEP skip}
  'Statements(s) to execute repeatedly
NEXT variable
```

The optional skip number tells IWBASIC to count by a certain number. One NEXT statement is needed for each FOR statement. To better demonstrate look at this example:

```
OPENCONSOLE
DEF counter:INT
DEF name$:STRING
INPUT "Enter your name ",name$

FOR counter = 1 TO 20 STEP 2
  PRINT "Hello ",name$
NEXT counter

PRINT "Press Any Key To Close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Will print a greeting 10 times. The *STEP 2* creates a loop that counts by twos until the counter variable is greater than or equal to 20. Once the counter has exceeded the end condition the loop ends. It is important to note that the loop includes the TO value.

To end a FOR/NEXT loop early use the [BREAKFOR](#) or [BREAK](#) command.

DO and WHILE statements

The DO statement executes one or more statements in your program at least once and repeats them until the condition in the UNTIL statement is met. The DO statement has the syntax of:

```
DO
  'Statement(s) to execute until condition = TRUE
UNTIL condition
```

The WHILE statement executes one or more statements in your program as long as the condition is met. The end of the loop is determined by the END WHILE statement. The WHILE statement has the syntax of:

```
WHILE condition
  'Statement(s) to execute while condition = TRUE
END WHILE
```

You can also use the pseudonym WEND in place of END WHILE.

If you examine the two statements, you will see that they are exact opposites of each other. The DO statement loops while a condition is false, the WHILE statement loops while a condition is true.

```
OPENCONSOLE
DEF x as INT
x=20
```

```

DO
  PRINT x
  x=x-1
UNTIL x < 11

WHILE x < 21
  PRINT x
  x=x+1
END WHILE

PRINT "Press Any Key To Close"

DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END

```

FOR EACH statement

A FOR EACH statement iterates through a linked list. It is fully documented in [Using linked lists](#).

8.8 Subroutines / Functions

Subroutines are sections of programs that need to be executed numerous times. While you could use [GOTO](#) to jump into and out of a section of code, it would be very hard to manage. Subroutines can be called from anywhere in the program and return execution from where they were called. Functions, commands, statements, API, procedure, and handlers are all common names for a subroutine. The distinction of the name depends on the usage.

All subroutines must begin with the [SUB](#) statement and end with the [ENDSUB](#) statement. Subroutines can appear anywhere in your source files and do not affect the path of execution until you call or [GOSUB](#) to them. Subroutines do not need to be declared if they are only used in the source file they are defined in. You must declare and external if you wish to use a subroutine located in another source file, this will be covered shortly.

In the SUB statement list any parameters that will be passed to the subroutine and also whether or not your subroutine returns a value. Here is a short example subroutine

```

SUB MyFirstSub( num as INT), INT
  DEF temp as INT
  temp = num * 4
  RETURN temp
ENDSUB

```

To use the cdecl calling convention specify the CDECL keyword.

```

SUB cdecl MyFirstSub( num as INT), INT

```

The variables defined in your subroutine, or defined in the SUB statement are called *local*

variables. A local variable is only accessible while your subroutine is being executed. Once control is returned to the code that called the subroutine the local variables do not exist. All local variables are stored in a special area of memory called the *stack*.

The stack size is set by the 'Advanced' tab on the *Executable Options* dialog or the *Project Options* dialog. The default stack commit size is set to 32K and indicates how large the size of the local variables in any one subroutine can be. If you exceed this size the compiler will generate a warning message so you can adjust as needed.

The [RETURN](#) statement can be omitted from the subroutine if the subroutine does not return a value. IWBASIC will insert the return statement when it encounters the ENDSUB statement.

```
SUB MySecondSub( num as INT)
  DEF temp as INT
  temp = num * 4
  PRINT temp
  'A RETURN not necessary here since the subroutine does not return a value and it
  'is the only point of return from the subroutine.
ENDSUB

SUB MyThirdSub( num as INT),INT
  DEF temp as INT
  temp = num * 4
  IF temp = 16 THEN RETURN temp
  RETURN -1: 'this return needed since the function returns a value.
ENDSUB
```

A subroutine's definition can wrap to multiple lines. The only requirement is that the "_" character must be the last in a line, not even space is allowed.

```
sub multiline( /*comments go here*/_
int a,_
int b)
print a,b
endsub
```

Calling the subroutine

Once a subroutine has been defined you can call it either by just using its name, or by using the GOSUB command. GOSUB is included for backwards compatibility with other versions of BASIC.

```
'call our function
a = MyFirstSub(23)
PRINT a
```

If a subroutine does not have any parameters use an empty parenthesis '()' to call it. Or alternatively use GOSUB

```
AnotherSub( )
GOSUB AnotherSub
```

Passing parameters

The above example shows passing a value to a subroutine. Variables can be passed by either *value* or *reference* depending on the type of variable and whether or not the BYREF keyword is used. When a variable is passed by reference the subroutine is allowed to make changes to the contents of the passed variable. When a variable is passed by value a copy of the contents of the variable is sent to the subroutine and the original variable will remain unchanged.

IWBASIC defaults to passing by value for all numeric types and passing by reference for strings, UDT's, etc. The following table lists the defaults:

TYPE	DEFAULT PASSED BY
INT/UINT	VALUE
INT64/UINT64	VALUE
FLOAT	VALUE
DOUBLE	VALUE
WORD	VALUE
CHAR	VALUE
FILE	REFERENCE
BFILE	REFERENCE
MEMORY	REFERENCE
STRING/WSTRING	REFERENCE
Arrays	REFERENCE
POINTER	REFERENCE
COMREF	REFERENCE
ANYTYPE	REFERENCE
UDT /WINDOW/ DIALOG	REFERENCE

Numeric types may be passed by reference by using the BYREF keyword or by using a POINTER parameter

```
SUB NewSub(a as UINT BYREF, b as FLOAT BYREF)
    'the variables can be modified and
    'will change the variables passed
    a = a * 50
    b = b + 2.0
RETURN
ENDSUB
```

Passing Arrays

Arrays require special handling when passing them to a subroutine. The subroutine needs to know the dimensions of the array at compile time. For single dimensioned arrays it is not necessary to supply a dimension and an empty bracket set will suffice '[]'. All arrays are passed by reference so anything done to the array in the subroutine will modify the array passed to it.

```

DEF myarray[10,50] as INT
DEF floatarray[10] as FLOAT
ArraysAreFun(myarray, floatarray)
END

SUB ArraysAreFun(iArray[10,50] as INT, flArray[] as FLOAT )
'do something with the arrays
RETURN
ENDSUB

```

Optional Parameters

Optional parameters at the end of the parameter list may be specified using the *OPT* keyword. A default value may be specified if the parameter is not included in the calling list. If a default parameter is not included then either 0 or a NULL string will be passed to fill in the optional parameter.

```

PrintIt("This string is centered",0,1)
PrintIt("This string is not",2)
DO:UNTIL INKEY$ <> ""
END

SUB PrintIt(str as STRING, row as INT, OPT center=0 as UINT)
  IF center = 1
    LOCATE row, (80 - LEN(str)) / 2
  ELSE
    LOCATE row, 0
  ENDIF
  PRINT Text$,
RETURN
ENDSUB

```

Declaring subroutines

You must use the [DECLARE](#) statement to use a subroutine defined in another source file in project mode, or if you wish to specify the C calling convention. The C calling convention is normally used when interfacing with C or other external libraries that require it. To use a subroutine located in a different source file use the EXTERN keyword with DECLARE:

```

DECLARE EXTERN OtherFilesSub(c as CHAR, b as INT), STRING
A$ = OtherFilesSub("A",24)

```

To use the cdecl calling convention in either a local or external declare specify the CDECL keyword

```

DECLARE CDECL MySub(c as CHAR, b as INT),STRING
DECLARE CDECL EXTERN _printf(format as STRING,...)

```

You can also use the DECLARE statement for subroutines located in the same file, but it is not necessary with IWBASIC. When the DECLARE statement is used locally it will override the

parameter names specified in the SUB statement so be sure to name them the same to avoid confusion

```
DECLARE AFunction(bb as DOUBLE),DOUBLE
...
SUB AFunction(bb as DOUBLE), DOUBLE
    bb = ASIN(bb) * ACOS(bb)
RETURN bb
ENDSUB
```

Global Subroutines

To allow other source modules to use a subroutine it must be declared as global by one of two methods.

First by using the GLOBAL keyword when creating the subroutine

```
GLOBAL SUB _gMySub(bb as FLOAT),FLOAT
    RETURN (bb * 1000.0) / 1000.0
ENDSUB
```

And then using DECLARE EXTERN in the other source modules where it will be used:

```
DECLARE EXTERN _gMySub(bb as FLOAT),FLOAT
```

To use the cdecl calling convention specify the CDECL keyword.

```
GLOBAL SUB cdecl _gMySub(bb as FLOAT),FLOAT
```

All global subroutines must be uniquely named or linking your executable will fail with a duplicate definition. When debugging programs only subroutines declared as global will display in the context window when a breakpoint is encountered. You will still get the correct file name and line number, just the name of the subroutine will default to the last global name in the path of execution.

The other method is demonstrated here:

In the header file:

```
declare extern mysub1()
```

in the source file:

```
$include "header.inc" ' with declare extern mysub1
sub mysub1() ' "global" isn't required if "extern" used.
```

There will be no "duplicate definition" error, and the SUB's will be automatically marked as global. If the compiler sees a function defined as external, and then a subroutine with the same name as the external declaration, it assumes the subroutine is to be shared with other modules. All the basic checks are made: the CDECL flag and the number and types of parameters must match.

Indirectly calling subroutines

IWBASIC supports indirectly calling subroutines through function pointers. A function pointer is a UINT variable that contains the address of a subroutine. The subroutine can be local or global. A common use for this is to create an array of subroutines, that all accept identical parameters, to be called by an index. The DECLARE statement needs to be used to set up a parameter template.

```
DECLARE fnTemplate(param as UINT),INT
```

You can use any valid name you wish for the template. After a subroutine is defined you can get the address of that subroutine using the & operator.

```
SUB Addition(param1 as FLOAT, param2 as FLOAT), FLOAT
    RETURN param1 + param2
ENDSUB

SUB Subtraction(param1 as FLOAT, param2 as FLOAT), FLOAT
    RETURN param1 - param2
ENDSUB

DEF fnArray[2] as UINT
fnArray[0] = &Addition
fnArray[1] = &Subtraction
```

Call the subroutine indirectly by using the ! operator, supplying the template name, variable containing the subroutine address, and any parameters

```
PRINT !<fnTemplate>fnArray[0](1.0, 3.0)
```

The DECLARE statement is only used as a template and will not look for a matching SUB when used as the parameter template for indirectly calling a subroutine. For a complete example see sample program: *indirect_functions.iwb*

Subroutines with a variable number of arguments

IWBASIC supports subroutines with a variable number of arguments. To specify a variable argument use the ellipses (...) after the first static parameter in the subroutines parameter list. One real parameter is required. Each additional argument sent to the subroutine can be accessed using a pointer.

```
SUB AddItUp(num as INT, ... ), FLOAT
```

In order to access the unknown number of arguments use the VA_START function to get a pointer to the first variable argument. VA_START requires the name of the last actual parameter.

```
DEF pParam as POINTER
pParam = VA_START( num )
```

Access the variable using any dereferencing operator

```
flTemp = #<FLOAT>pParam
```

Use pointer math to retrieve successive arguments

```
pParam+=4  
flTemp2 = #<FLOAT>pParam
```

The variable arguments are passed without regard to type and size information. It is up to the programmer to determine the size of each argument sent to the subroutine. A common method is to use an index variable or formatting string. For example The [USING](#) function uses a variable number of arguments whose type and size depends on a formatting string.

All numeric variable types are passed by value. To keep inline with industry standards all floating point types are converted to a `DOUBLE` before being passed. Strings and UDT's are passed by reference. Variable sizes are listed in the [Variables](#) topic. References to strings and UDT's take up four bytes.

For a complete example see the sample file *variable_arguments.iwb* included with the distribution.

Labels

Labels defined inside a subroutine are private to that subroutine. This means the same label can be used in multiple subroutines (eg. [LABEL](#) cleanup). [GOTO](#) can not be used to jump into or out of a subroutine Internally all local labels have unique names.

```
label outside  
  goto cleanup <- undefined, unless defined in global namespace  
  goto subroutine1 <- error  
  
sub subroutine1()  
  goto outside <- undefined  
  goto cleanup  
cleanup:  
  '...  
endsub  
  
sub subroutine2()  
  goto outside <- undefined  
  goto cleanup  
cleanup:  
  '...  
endsub
```

Special return values

There are two special return types: `FPU` and `__m128`. The types are not defined as language keywords. They are recognized by the compiler only when they are used as return types. The user must define them with before using them.

The `FPU` type marks a function as returning a float/double in the ST0 FPU register (for Bass users).

```
type FPU
endtype

declare GetFpu(), FPU

print GetFpu()
end

$asm
align 4
GetFpu:
fld qword [mydata]
ret

segment .data
mydata:
dq 1.23456789
$endasm
```

The `__m128` type marks a function as returning an array of 4 floats or two doubles in XMM0 register (SSE users). Because the data is 128-bit, it is copied to temporary heap memory, in the same way that a structure or string would be returned.

```
union __m128
float m128_f32[4]
float f32[4]
double d64[2]
endunion

declare get_xmm_value(), __m128

__m128 v = get_xmm_value()

print v.m128_f32[0], v.f32[1], v.f32[2], v.f32[3]
end

$asm
align 4
get_xmm_value:
movups xmm0, [myvalue]
ret

segment .data
align 16
myvalue:
dd 1.0, 2.0, 3.0, 4.0
$endasm
```

8.9 Using Linked Lists

A linked list is a data structure used to maintain a dynamic series of data. Think of a linked list as a line of elephants in a circus where each elephant is holding on to the tail of the next elephant. If you know where the first elephant is, you can follow it's trunk to the next one. By following trunks, you

can find any elephant in the chain. When you get to an elephant that isn't holding on to another elephant's tail, you know you are at the end.

IWBASIC supports generic linked lists through dedicated functions and a variation of the FOR statement. Each element of the list contains a data pointer for your use in storing any type of dynamic variable created with the NEW function.

Creating a new list

Before you can add data to a list you must first create a blank list with the [ListCreate](#) function. ListCreate returns a pointer to the new list ready for use. Assign the return to a pointer variable.

```
DEF myList as POINTER
mList = ListCreate( )
```

Adding data to the list

After the list is successfully created add data to it using either the [ListAdd](#) or [ListAddHead](#) function. List add places the new data node at the end of the list while ListAddHead places the new data node at the beginning of the list, moving the current head down one position.

Both functions require two parameters. The pointer variable containing the list and a pointer to the new data to store in the list. For convenience they return the data pointer.

The following are equivalents:

```
tempData = NEW(INT,1)
ListAdd(myList, tempData)
```

OR

```
tempData = ListAdd(myList, NEW(INT,1) )
```

Initialize the newly added data using standard pointer operations

```
#<INT>tempData = 5
```

Iterating a linked list

Iterating through a linked list means reading each element of the list one by one until the desired data is found or the end of the list is reached. IWBASIC provides two methods for iteration, a simple and an advanced. The simple method uses the FOR EACH statement and loops through the entire list.

```
FOR tempData = EACH myList AS INT
PRINT #tempData
#tempData += 5
```

```
NEXT
```

Note that the AS keyword is optional and performs an automatic [SETTYPE](#) on the returned pointer. This alleviates the need to use type casting when accessing the data. Without the AS keyword the loop would look like:

```
FOR tempData = EACH myList
  PRINT #<INT>tempData
  #<INT>tempData += 5
NEXT
```

For advanced accessing of the elements use [ListGetFirst](#), [ListGetNext](#) and [ListGetData](#)

```
pos = ListGetFirst(myList)
WHILE pos <> 0
  tempData = ListGetData(pos)
  PRINT #<INT>tempData,
  pos = ListGetNext(pos)
ENDWHILE
```

ListGetFirst returns a pointer to the first node in the list. ListGetNext returns the pointer to the next node in the list after the passed node or NULL if the end of the list has been reached. ListGetData returns the data pointer contained in the element passed.

Removing elements while iterating

Removing nodes from a list while iterating is accomplished by using advanced iteration and the [ListRemove](#) function.

```
pos = ListGetFirst(mylist)
WHILE pos <> 0
  tempData = ListGetData(pos)
  IF #<INT>tempData = 5
    pos = ListRemove(pos, TRUE)
  ELSE
    pos = ListGetNext(pos)
  ENDIF
ENDWHILE
```

ListRemove returns a pointer to the next node in the list after the deleted node or NULL if the end of the list has been reached. The TRUE specifies to also delete the data pointer with the DELETE function. It is important to remember that ListRemove takes the place of ListGetNext in this instance.

Removing all elements and deleting the list

To remove all elements from a linked list use the [ListRemoveAll](#) function.

```
ListRemoveAll(myList, TRUE)
```

Once a list has been deleted with ListRemoveAll you must not use the list pointer again unless re-initialized with ListCreate. Specifying TRUE for the bDelete parameter deletes all of the data items

as well. If you specify FALSE then it is your responsibility to delete your data by other means.

8.10 Using DATA Statements

IWBASIC supports DATA statements through named data blocks. Any number of separate data blocks can be used in your program and there is no limit to the number of data items contained within a block.

Defining a block of data

The normal place to define a data block is at the end of your source file. Although they can appear anywhere. Begin the definition with the [DATABEGIN](#) statement and end with the [DATAEND](#) statement. Define the data items between the block using the [DATA](#) statement.

```
DATABEGIN mydata
DATA "monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sunday"
DATA 1, 2, 3, 4, 5, 6, 7
DATAEND
```

The DATA statement accepts *strings*, *Unicode strings*, *integers*, *floats* and *double precision* literals. Variables are not allowed as the compiler has to resolve the data statement at compile time. DATA is limited to 100 items per line, separate your data for readability. The same basic math operations used for creating constants can be used when defining a data statement.

Reading data

Data can be read by your program using the [GETDATA](#) function. GETDATA requires the name of the data block and a variable to store the read data into. The compiler will handle conversions between variable types so you can specify a float variable for double precision data if needed.

```
DEF str as STRING
DEF iDay as INT
FOR x = 1 to 7
    GETDATA mydata, str
    PRINT str
NEXT x
FOR x = 1 to 7
    GETDATA mydata, iDay
    PRINT iData
NEXT x
```

Each use of GETDATA advances an internal pointer to the next data item in the list. There is no end of data marker so it is possible to read past the end of the data block. A common programming technique is to create your own marker using either a special number or an empty string for string data..

```
DEF sName as STRING
DO
```

```

    GETDATA limitdata, sName
    IF sName <> "" THEN PRINT sName
UNTIL sName = ""

DO:UNTIL INKEY$ <> ""
END

DATABEGIN limitdata
DATA "Fred", "James", "Gary", "Timmy", "Paul", ""
DATAEND

```

Restoring the data pointer

You can move the data pointer in the data block back to the beginning by using the [RESTORE](#) command

```
RESTORE mydata
```

After the restore GETDATA will read from the beginning of the data block.

8.11 Using DLL's and the Windows API

IW BASIC supports using external DLL's to extend the capabilities of the language through the use of import libraries. An import library is a special .LIB file that contains the names and entry points of all of the functions contained within a particular DLL. The Windows API is a collection of functions in system DLL's located in either the windows\system or windows\system32 directory.

IW BASIC includes import libraries for many of the standard Windows API DLL's. All import libraries are located in the *libs* directory in the main installation directory for IW BASIC.. See the list at the end of the topic.

An include file containing declares, types and constants for much of the Windows API is included with your installation. All Win API function names are aliased with a leading underscore and to remove the 'A' on ANSI function names. Example:

```

$INCLUDE "windows.inc"
DEF ms AS MEMORYSTATUS
ms.dwLength = LEN(MEMORYSTATUS)
GlobalMemoryStatus(ms)
PRINT "Free Memory:", USING("-#,#####&",ms.dwAvailPhys," Bytes ")
DO:UNTIL INKEY$ <> ""

```

Creating import libraries

To create an import library for a DLL select the *Tools* menu and select *Create Import Library*. Browse to the DLL you wish to use in your programs and click on the *Open* button. The new import library will have a .lib extension with the same name as the DLL and will be copied automatically to the *libs* directory. You only need to do this once for a new DLL. Once the import library is added you can use it in any program.

Including the import library in the build

After successfully creating an import library for a DLL you need to tell the IW BASIC Linker that you wish to include it when compiling programs. Use the \$USE preprocessor command to include linker and import libraries.

```
$USE "mydll.lib"
```

You may also specify a full path name.

```
$USE "c:\\mylibs\\mydll.lib"
```

You do not need to use the \$USE command to access any of the functions available in the default Windows API DLL's. The linker automatically includes the import libraries for the DLL's listed at the end of the topic.

Calling functions in the DLL

To use a function in a DLL it must first be declared using the IMPORT keyword. The IMPORT keyword tells the compiler that the function is located in either a DLL or import library.

```
DECLARE IMPORT, GetSysColor(nIndex as INT),UINT
color = GetSysColor(5)
```

Note the comma in the DECLARE statement. You can *alias* function names to resolve naming conflicts or to declare the same function more than once with different parameter types.

```
DECLARE IMPORT, GetColor ALIAS GetSysColor(nIndex as INT),UINT
color = GetColor(0)
```

Functions that use the C calling convention will usually be noted in the documentation for the DLL and can be used by including the CDECL keyword. The Windows API function `wsprintf` uses the C calling convention.

```
DECLARE CDECL IMPORT, wsprintfA(buf as STRING, format as STRING, ... ), INT
DEF out as STRING
wsprintfA(out, "The number is %d", 99)
PRINT out
```

Calling functions in the C runtime library

The C runtime DLL uses the import library `crtdll.lib`. However it is linked differently from other import libraries. To use the functions contained within the C runtime DLL use the EXTERN keyword instead of the IMPORT keyword.

```
DECLARE CDECL EXTERN _sprintf(buf as STRING, format as STRING, ...),INT
```

All functions in the C runtime library require the CDECL keyword and usually begin with an underscore.

Calling functions with C name mangling

Certain C and C++ compilers create DLL's with what's known as *mangled function names*. The format of the mangling varies from compiler to compiler but is generally the function name with a beginning underscore and a trailing @ symbol with a number. The number is used by other linkers to determine how many parameters are pushed on the stack. The DLL's creator can map the function names to normal ones but in most cases the name mapping is not done.

To use DLL functions with mangled function names you must use the EXTERN keyword instead of the IMPORT keyword. For example if a C DLL was created with a function named INIT_the_system(bInit as INT),INT then the actual name exported in the DLL would be:

```
_INIT_the_system@4
```

The correct DECLARE would look like:

```
DECLARE EXTERN _INIT_the_system@4(bInit as INT),INT
```

Importing variables

Variables can be imported from dlls. The following is an example of how it is accomplished.

```
import __argc as int
import __wargv as pointer ' call __wgetmainargs to initialize
import __argv as pointer ' call __getmainargs to initialize

$use "msvcrt.lib"
declare cdecl import, __getmainargs(pointer argc, pointer pargv, pointer penv, int e
declare cdecl import, __wgetmainargs(pointer argc, pointer ppwargv, pointer ppwenv, in

int    argc, new_mode, a
pointer pargv, penv
pointer pwargv, pwenv

' initialize __argc and __argv
__getmainargs(&argc, &pargv, &penv, 0, &new_mode)
' initialize __argc and __wargv
__wgetmainargs(&argc, &pwargv, &pwenv, 0, &new_mode)

print "you passed ", __argc-1, "arguments"
for a=0 to __argc-1
  print " arg ", a, "=", *<string>(*<pointer>__argv[a])
  print "warg ", a, "=", *<wstring>(*<pointer>__wargv[a])
next a
```

Compile as *varImport.exe* (console target). Then open a command prompt window to the parent folder, and type:

```
varImport 123 abc "z:\program files"
```

Import libraries included with IWBASIC as of Version 2.0

kernel32.lib
user32.lib
gdi32.lib
comdlg32.lib
comctr32.lib
shell32.lib
winmm.lib
ole32.lib
olepro32.lib
oleaut32.lib
winspool.lib
shlwapi.lib
uuid.lib
ddraw.lib
dinput.lib
advapi32.lib
crtdll.lib
ddraw.lib
dinput.lib
dsound.lib
odbc32.lib
rasapi32.lib
ws2_32.lib
wsock32.lib

The list of import libraries included with the distribution is subject to change. To be sure check the contents of the libs directory. Other import libraries internal to the use of the compiler are not listed here.

8.12 Conditional Compiling

Conditional compiling simply stated is excluding or including portions of code from your source file based on one or more conditions. Similar to a conditional statement controlling the flow of execution of program statements.

The preprocessor is a part of the compiler that scans your source code before the actual compiling to assembly language begins. Among many other tasks the preprocessors looks for conditional compiling statements to determine what should be included in the final executable. This is done by defining conditional identifiers that can be tested anywhere in your code.

Creating a conditional identifier

A conditional identifier is a word or sequence of letters and numbers that you want to define as

TRUE to the preprocessor. Think of them as a constant without any real value. They are only visible to the preprocessor and are created with the \$DEFINE \ #DEFINE statement. For example suppose you have two versions of your program, a trial version and a paid for version. Maintaining two separate projects would be a solution but its much easier to write the source so you can quickly compile one or the other.

```
$DEFINE TRIAL_VERSION
```

Would define the identifier TRIAL_VERSION as TRUE to the preprocessor.

The \$DEFINE \ #DEFINE statement can also be used to create constants. See [Constants and Literals](#) for details.

Excluding or including code to be compiled

Using the preprocessor commands \$IF, \$IFDEF, \$IFNDEF, \$ELSE, \$ELIF and \$ENDIF you can test for a conditional identifier and include or exclude any section of code necessary. Using the identifier from above:

```
$IFDEF TRIAL_VERSION
    ShowRegisterDialog( )
    Save_Allowed = FALSE
$ELSE
    ShowWelcomeDialog( )
    Save_Allowed = TRUE
$ENDIF

$if version == 1
... something
$elif version == 2
... something
$endif
```

The following operators are available for use: =, ==, <>, !=, +, -, *, /, %, |, or, || (xor), &, &&, and, <<, >>, !, ~, >, >=, <, <=

```
$if (1=2) or (MYVERSION < 1)
$error "i'm sorry, but something here is wrong"
$endif

$if (VERSION > 0x0202) and (SUBVERSION <> 0)
```

It is important to note that the preprocessor statements are not actually program statements but control what the compiler processes. In our hypothetical trial version/full version code above if the identifier TRIAL_VERSION has been defined then a registration dialog would be shown and saving files disabled. The code to show the welcome dialog is skipped entirely and would not generate any machine code in the executable.

Conditional compiling statements can be nested to allow for advanced situations. Continuing with

our hypothetical trial version and we wanted to maintain debugging code in the source:

```
$IFDEF TRIAL_VERSION
    ShowRegisterDialog( )
    Save_Allowed = FALSE
    $IFDEF DEBUG
        MESSAGEBOX win1,key$, "Key"
    $ENDIF
$ELSE
    ShowWelcomeDialog( )
    Save_Allowed = TRUE
    $IFDEF DEBUG
        DEBUGPRINT "Program started: Registered name = " + name$
    $ENDIF
$ENDIF
```

For situations where you only need a negative test just use \$IFNDEF

```
$IFNDEF DEBUG
    ShowWelcomeDialog( )
$ENDIF
```

\$ELIFDEF (ElseIfDefined) and \$ELIFNDEF (ElseIfNotDefined) are included to complete the set of available conditional commands.

Notes on conditional compiling

The \$DEFINE \#DEFINE statement is source file level meaning that if your using a project the identifier is only defined in the source file the \$DEFINE \#DEFINE statement appears in. To define a preprocessor identifier across the entire project use an include file and \$INCLUDE that file at the beginning of every source file.

Everything is skipped if the condition is false. This includes variable definitions, DECLARE statements, etc. If your program depends on a variable or DECLARE statement regardless of the condition then be sure to define them outside of the \$IFDEF block.

\$IFDEF and \$IFNDEF can also check for the existence of a constant.

8.13 \$INCLUDE Command

The \$INCLUDE preprocessor command allows reading header files and even other source files (including assembly code *.asm files) into the source file being compiled. Header files have a .inc extension in IWASIC and are traditionally used for API definitions, constants, TYPE definitions, and COM declarations.

The \$INCLUDE command expects a string containing the name of the file to be included. The string is not a constant and the normal rules for specifying paths do not apply. If a full path is not specified the *include* directory will be searched for the file. The include directory is located just off

the main installation directory for IW BASIC, which if you chose defaults during installation would be:

C:\Program Files\EBDev\Include\

You are free to store any include file there you wish to aid in developing programs. The IDE knows what include files are and after creating them choose *Save As..* from the *File* menu and choose "IW BASIC Include Files (*.inc)" from the drop down file types box.

Example include statements:

```
$INCLUDE "ishelllink.inc"  
$INCLUDE "c:\myIncludes\bitmaps.inc"
```

As noted above the quoted string is not a constant and does not need the double backslash in the paths like a constant string would in your programs.

The search order for searching files accessed by \$INCLUDE is:

1. The directory of currently parsed file
2. The Include directory
3. User directory list added in the [/i command lines switch](#) or "[IncludePath](#)" option
4. The Bin directory
5. The list from INCLUDE environment variable.

The format is same as in PATH variable: "c:\dir1;d:\dir2;e:\dir2\dir 5; ... z:\last dir"

Notes, caveats, warnings

Any errors in an include file will correctly show the file and line number in the *build* window.

All include statements should be placed as the first lines in your source file. The contents of the included file are inserted at the point the \$INCLUDE statement appears.

While you can include other source files containing subroutines keep in mind the above warning. The \$INCLUDE statement is a preprocessor command and the compiler will attempt to inject and compile anything in the included file at the point of insertion. For example this would be a bad idea and will result in many errors:

```
IF something  
$INCLUDE "file.iwb"  
ENDIF
```

The correct method is to use conditional compiling at the beginning of your source file.

```
$IFDEF SHAREWARE  
    $INCLUDE "sharewaredefs.inc"  
$ELSE  
    $INCLUDE "fulldefs.inc"  
$ENDIF
```

Include file may be nested up to 100 levels deep. That is one include file can include others recursively up to 100 times. It is very unusual to have more than two levels of include file nesting though.

8.14 Using COM

The Component Object Model (COM) is a platform-independent, distributed, object-oriented, system for creating binary software components that can interact. COM is the foundation technology for OLE, ActiveX, as well as others.

In laymen's terms a COM object represents a collection of subroutines that can be created at will. Kind of like a DLL. The heart of COM is the interface which describes to the compiler what subroutines are available in a particular object.

IWBASIC supports COM at a basic level with functions for describing interfaces, calling members of objects, and dealing with GUID's (Globally Unique IDentifiers). This is a topic for advanced programmers and there are a great many books and online texts dedicated to interfacing with objects which will cover it in greater detail.

The interface

At the heart of COM is the interface. The interface defines what functions, known as methods, are callable by your program. Similar to declaring subroutines, a method definition gives the name of the function and its parameters. To begin definition of the interface use the [INTERFACE](#) keyword and end the definition with the [ENDINTERFACE](#) keyword. The only statements allowed between the INTERFACE/ENDINTERFACE block are [STDMETHOD](#) statements and source comments.

The INTERFACE command expects one parameter, the identifier of the interface. For example suppose we want to access the Windows interface for creating shortcuts, also known as links. The interface is known as IShellLink and there are both ASCII and UNICODE versions for your use. We will define the ASCII version for brevity.

```
INTERFACE IShellLinkA
```

After the INTERFACE statements comes the method declaration. This is the hard part as most documentation for methods on-line or in books will be for C or C++. And almost all of them omit the first three methods because they are actually part of a parent interface known as IUnknown. Namely QueryInterface, AddRef and Release. You can assume they are there but to be sure check the documentation for the COM object you are trying to use.

```
/** IUnknown methods */
STDMETHOD QueryInterface(riid as POINTER, ppvObj as POINTER)
STDMETHOD AddRef( )
```

```
STDMETHOD Release( )
```

With STDMETHOD definitions there is only one return type, the documentation for COM calls it an HRESULT, its really just an INT. It is automatically the standard return type for methods and does not need to be specified. Continuing with the interface definition for IShellLink:

```
/**/ IShellLinkA methods ***/
STDMETHOD GetPath(pszFile as STRING,cchMaxPath as INT, pdf as WIN32_FIND_DATA, fFlags
STDMETHOD GetIDList(ppidl as POINTER)
STDMETHOD SetIDList(ppidl as LPCITEMIDLIST)
STDMETHOD SetDescription(pszName as STRING,cchMaxName as INT)
STDMETHOD SetDescription(pszName as STRING)
STDMETHOD GetWorkingDirectory(pszDir as STRING,cchMaxPath as INT)
STDMETHOD SetWorkingDirectory(pszDir as STRING)
STDMETHOD GetArguments(pszArgs as STRING,cchMaxPath as INT)
STDMETHOD SetArguments(pszArgs as STRING)
STDMETHOD GetHotkey(pwHotkey as WORD BYREF)
STDMETHOD SetHotkey(wHotkey as WORD)
STDMETHOD GetShowCmd(piShowCmd as INT BYREF)
STDMETHOD SetShowCmd(iShowCmd as INT)
STDMETHOD GetIconLocation(pszIconPath as STRING,cchIconPath as INT,piIcon as INT BYREF)
STDMETHOD SetIconLocation(pszIconPath as STRING,iIcon as INT)
STDMETHOD SetRelativePath(pszPathRel as STRING,dwReserved as UINT)
STDMETHOD Resolve(hwnd as UINT,fFlags as UINT)
STDMETHOD SetPath(pszFile as STRING)
ENDINTERFACE
```

An interface may need other data besides the methods. In our example two of the methods require UDT's of WIN32_FIND_DATA and ITEMIDLIST. The complete interface definition is included with IW BASIC in the *include* directory and can be used in your code by using `$INCLUDE "ishellink.inc"` in the source file you wish to use the IShellLink object from.

Creating the object

To create and use COM objects your program must first tell Windows that it wants to use COM, and then create the object. The standard command sets includes DECLARE's for the necessary API functions CoInitialize, CoUninitialize and CoCreateInstance. Some interfaces have dedicated API's for creating the object, again consult any necessary documentation. The pointer to the interface, once created, is stored in a special variable type called COMREF. You must define a variable of type COMREF for every interface you wish to access.

For our example these are the steps necessary to initialize COM and create the object:

```
DEF iShell as COMREF
DEF result as INT
CoInitialize(NULL) /* make sure COM is initialized */
'Create the object
result = CoCreateInstance(_CLSID_ShellLink, NULL, CLSCTX_INPROC_SERVER, _IID_IShellLink)
'Always check the result. If equal to zero then its OK to proceed
IF result = 0
    'use the interface
ENDIF
```

```
CoUninitialize( ) /* Tell Windows we are done with COM */
```

Calling methods in the object

Once a valid object is created you can call any method in the object using the COM method operator '->'. First you need to tell the compiler what interface to use with created object. This is how we link objects and interfaces together. Use the [SET_INTERFACE](#) command to instruct the compiler what set of methods can be called from a COMREF variable.

```
SET_INTERFACE iShell, IShellLinkA
iShell->SetPath("C:\\Projects\\MyProg.exe")
iShell->SetDescription("Its my program")
```

Think of the COM method operator as "From this interface -> call this function".

When you are done using an object you must always call its Release method. This deletes the object and frees any memory used by it.

```
iShell->Release( )
```

Once an object is released you must not attempt to call any of the objects methods without recreating the object first. Doing so will end your program quickly with an Access Violation.

NOTE: As of version 1.06 of the core library you can directly use an interface name when defining a COM object. This eliminates the need for SET_INTERFACE. However the command is maintained for backwards compatibility. Example:

```
DEF iShell as IShellLinkA
DEF result as INT
result = CoCreateInstance(_CLSID_ShellLink, NULL, CLSCTX_INPROC_SERVER, _IID_IShellLinkA)
'Always check the result. If equal to zero then its OK to proceed
IF result = 0
    iShell->SetPath("C:\\Projects\\MyProg.exe")
    iShell->SetDescription("Its my program")
ENDIF
```

Interfaces within interfaces

Interfaces can and will contain other interfaces internally. We mentioned the IUnknown methods earlier and the first method in our IShellLinkA interface was not surprisingly QueryInterface. QueryInterface gives access to other interfaces that an object might contain. Our IShellLink interface contains once called IPersistFile that allows the object to save itself to disk. In our example the shell link itself is saved to a file to create the shortcut. Similar to CoCreateInstance, QueryInterface takes a GUID and returns a COMREF.

```
DEF ppf as COMREF
result = iShell->QueryInterface(_IID_IPersistFile, ppf)
IF result = 0
    SET_INTERFACE ppf, IPersistFile
    ppf->Save(wsz, TRUE)
    ppf->Release( )
```

```
ENDIF
```

GUID usage

A GUID or Globally Unique Identifier is a special UDT that contains the numeric identifiers needed for accessing COM objects among others. We have been using them all along in our examples without specifically defining them. IWBASIC links with the common system GUID's through the UUID.LIB library. All GUID's in this library can be used by defining them as EXTERN and they all start with a leading underscore, which may be different the the official Microsoft documentation. For example:

```
EXTERN _IID_IPersistFile as GUID
```

You can create your own GUID's by defining a UDT and using the [DEFINE_GUID](#) function

```
DEF myGuid as GUID
DEFINE_GUID( myGuid, 0xD7B70EE0, 0x4340, 0x11CF, 0xB0, 0x63, 0x00, 0x20, 0xAF, 0xC2, 0xCD, 0x35
```

Many third party COM objects include the GUID definitions in the documentation or include files. The DEFINE_GUID function closely resembles the C macro with the same name to aid in conversions.

See Also: The complete shell link example can be found in the sample file *shell_com_example.iwb* which uses the include file *ishelllink.inc* mentioned earlier.

8.15 Object Oriented Programming

Object Oriented Programming, or OOP as it is commonly known, is a programming technique that uses "objects" to design applications and computer programs. An object can be thought of as a collection of subroutines and related data encapsulated into a distinct package. Each subroutine in an object is referred to as a *method* and the data associated with the object as *members*.

To create an object the compiler must know what methods and member variables are included. In IWBASIC this is done by specifying a *class* definition. A class definition is similar to defining a UDT with the exception that you will also be providing the names of the methods for that class. Classes and UDT's are related in the fact that internally member variables of a class are stored in that same manner as members of a UDT.

Class Definitions

The CLASS statement begins the definition of a new class. A class definition is ended with the END CLASS statement. Each variable defined within the class definition specify the member variables that are part of the created object. Methods of the class are specified by using a DECLARE statement in much the same manner that you would declare a subroutine. Here is a short example:

```

CLASS employee
  'methods
  DECLARE CalculatePay(hours as float),float
  DECLARE SetPayRate(amount as float)
  DECLARE SetEmployeeID(id as int)
  DECLARE PrintPaycheck( )
  'members
  int m_employeeID
  float m_payrate
  float m_lastpay
  string m_name
END CLASS

```

In the example above we defined the class of an object named 'employee'. The member variables are prefixed with `m_` as a convenience for recognizing them as members later on and is a personal preference. The next step in creating a class is writing the method subroutines. This is known as the *implementation of a class*. A method subroutine is written in the same manner as a regular subroutine with a few important differences. The name of a method subroutine is comprised of two parts separated by a double colon:

```
SUB employee::CalculatePay(hours as float),float
```

On the left side of the double colon is the name of the class the method belongs to. On the right side of the double colon is the name of the method itself. A method subroutine follows the same syntax as a regular subroutine with some added benefits. For example the member variables of a class are directly accessible by the method subroutine. This is an important concept to learn as it is the heart of OOP. The *encapsulation* of related data and subroutines. Let's continue the writing of the method subroutines:

```

SUB employee::CalculatePay(hours as float),float
  m_lastpay = m_payrate * hours
  RETURN m_lastpay
END SUB

SUB employee::SetPayRate(amount as float)
  m_payrate = amount
END SUB

SUB employee::SetEmployeeID(id as int)
  m_employeeID = id
END SUB

SUB employee::PrintPayCheck( )
  PRINT "Pay to the order of " + m_name + " ***$" + USING("###.##",m_lastpay)
END SUB

```

Creating and using the Object

An object is created, or *instantiated*, by using it like any other variable type. You can create an object either locally, globally or by using the NEW statement. Using the example class above we

can create an instance of an employee object:

```
DEF emp as employee
```

The member variables of the object are accessed in the same way as a UDT, using the dot operator. The method subroutines of an object are also accessed using the dot operator:

```
emp.m_name = "John Doe"
emp.SetEmployeeId(1111)
emp.SetPayRate(7.25)
emp.CalculatePay(40.5)
emp.PrintPayCheck( )
```

When creating an object dynamically you must set the type of the pointer used. This is necessary for DELETE to work properly with classes and to call the class destructor:

```
POINTER pEmp
SETTYPE pEmp,employee
pEmp = NEW(employee,1)
#pEmp.m_name = "John Doe"
#pEmp.SetEmployeeId(1111)
#pEmp.SetPayRate(7.25)
#pEmp.CalculatePay(40.5)
#pEmp.PrintPayCheck( )
DELETE pEmp
```

Constructors and Destructors

A constructor is a method of an object that is executed when the object is created. Similarly a destructor is executed when the object is destroyed. Objects are destroyed when the DELETE command is used on dynamically allocated objects, or when the object goes out of scope for statically created ones. For example an object that is created statically in a subroutine will automatically be destroyed when the subroutine ends. An object that is created globally, outside of a subroutine, will be destroyed when the program ends.

The constructor is a method that has the same name as the class of the object and a destructor has the same name as the class of the object prefixed with an underscore. For example let's add a constructor and destructor method to our employee class:

```
CLASS employee
  'methods
  DECLARE employee( ) : 'The object constructor
  DECLARE _employee( ) : 'The object destructor
  DECLARE CalculatePay(hours as float),float
  DECLARE SetPayRate(amount as float)
  DECLARE SetEmployeeID(id as int)
  DECLARE PrintPaycheck( )
  'members
  int m_employeeID
  float m_payrate
```

```

float m_lastpay
string m_name
pointer m_history
END CLASS

```

Constructors are generally used to initialize the member variables of an object. This is an important step since an object that is created statically in a subroutine will have random data stored in the member variables. Constructors are also used to allocate any memory needed by the object.

```

'Constructor
SUB employee::employee( )
    m_employeeID = -1
    m_payrate = 0.0f
    m_lastpay = 0.0f
    m_name = "None"
    m_history = NEW(CHAR,4096)
END SUB

```

A destructor is commonly used to clean up any memory allocated by an object. In the above constructor we allocated memory for a 4K string, supposedly as a text description of the employees history. We can delete that dynamically created string in the destructor:

```

SUB employee::_employee( )
    DELETE m_history
ENDSUB

```

This is another powerful feature of OOP. The object manages its own data and cleans up after itself.

This also applies for classes defined in a structure (any level):

```

class myclass
    declare myclass()
    declare _myclass()
endclass

type child
    myclass c
endtype

type mytype
    myclass a
    child b
endtype

mytype t
end

sub myclass::myclass()
    print "constructor"
endsub

sub myclass::_myclass()
    print "destructor"
endsub

```

Output:

```

constructor
constructor
destructor

```

destructor

Access protection

It is generally considered bad form to directly access a member variable from outside of a method implementation. In the above examples we have been setting the name of the employee directly using a dot operator. While this might not seem like a bad thing, consider the dynamic string that is allocated in the constructor. If the program that uses your object sets `m_history` to `NULL` or tries to use it for a different purpose than a text description, it may undermine the intended operation of the class. Also there may be methods of your class that are only used by that class internally and should not be executed outside of the object.

To aid in limiting outside access to an objects data IWBASIC provides three keywords, `PUBLIC`, `PRIVATE` and `PROTECTED` that control how an objects methods and members may be used. By default all methods and members have `PUBLIC` access meaning they are accessible by both the object and the outside world that uses the object.

A `PRIVATE` method or member can only be accessed by a method of that object. A `PROTECTED` method or member can only be accessed by a method of that object, or any objects that are derived from that class. Derived classes will be covered shortly.

When one of the access protection keywords is encountered all of the methods and members following that keyword will have that access until the next access protection keyword is used. Let's change our employee class to control access to the methods and members:

```
CLASS employee
  'methods
  DECLARE employee( ) : 'The object constructor
  DECLARE _employee( ): 'The object destructor
  DECLARE CalculatePay(hours as float),float
  DECLARE SetPayRate(amount as float)
  DECLARE SetEmployeeID(id as int)
  DECLARE PrintPaycheck( )
  DECLARE SetHistory(history as STRING)
  DECLARE SetEmployeeName(name as STRING)
PRIVATE
  DECLARE ClearHistory( )
  'members
PROTECTED
  int m_employeeID
  float m_payrate
  float m_lastpay
  string m_name
  pointer m_history
END CLASS
```

In the changed class definition we have one method that is marked as `PRIVATE` to the class. This method is only executable from within another method of the same object. Attempting to execute the `ClearHistory` method from outside of the object will result in the compiler generating an error

message. All of the member variables are marked as PROTECTED which limits their accessibility to this class or any derived classes. Attempting to access a protected member will also generate a compiler error.

Inheritance

Inheritance is the technique of extending one class, known as a *base class*, to add additional functionality. The new class, known as a *derived class*, contains all of the methods and members of the base class. The terminology is important as the relationship between base and derived classes is often confused with a parent/child relationship. In fact, inheritance is an "is-a" relationship: manager *is a* type of employee.

IWBASIC supports single inheritance and the name of the base class is specified using the optional parameter of the CLASS statement. For example if we wish to extend the functionality of the employee class and define a new class called *manager* we don't have to duplicated all of the methods of the employee class. We simply have to derive from the employee class:

```
CLASS manager, employee
'Constructor
  DECLARE manager( )
  DECLARE SetDividend(amount as float)
  DECLARE SetBonus(amount as float)
  DECLARE CalculateManagerPay(hours as float)
PROTECTED
  float m_dividend
  float m_bonus
END CLASS

SUB manager::manager
  m_dividend = 0f
  m_bonus = 0f
END SUB

SUB manager::CalculateManagerPay(hours as float)
  m_lastpay = (m_payrate * hours) + m_bonus + (m_dividend * company_profit)
END SUB
```

The derived class contains all of the methods of the employee class and the methods declared in the manager class. When using the derived class nothing special has to be done to access the methods defined in the employee class. They are part of it.

```
DEF man as manager
man.SetEmployeeName("Jim Doe")
man.SetEmployeeId(1111)
man.SetPayRate(15.25)
man.SetDividend(.015f)
man.SetBonus(500f)
man.CalculateManagerPay(40.5)
man.PrintPayCheck( )
```

In the above class we have a constructor that zeros out the two member variables of the manager object. What about the constructor of the employee object? The compiler executes all

constructors when the object is created starting with the base class all the way up to the last derived class. Similarly when the object is destroyed all of the destructors are called in reverse order starting with the last derived class all the way down to the base class.

Method overriding and polymorphism

Method overriding is a technique used by the compiler to call the correct method in a derived class when a method of the same name exists in a base class. These methods are known as *virtual methods* and are declared with the VIRTUAL keyword. Consider our two classes defined so far, a manager is a type of employee but has a different method of calculating a paycheck. What if we had ten different classes derived from employee that all require a unique calculation. Coming up with a separate method name for each type of employee would soon become hard to manage. When using dynamically created classes it becomes even more difficult to know what kind of class the pointer refers to. We solve this problem by using virtual methods:

```
CLASS employee
  'methods
  DECLARE employee( ) : 'The object constructor
  DECLARE _employee( ): 'The object destructor
  DECLARE VIRTUAL CalculatePay(hours as float),float
  DECLARE SetPayRate(amount as float)
  DECLARE SetEmployeeID(id as int)
  DECLARE PrintPaycheck( )
  DECLARE SetHistory(history as STRING)
  DECLARE SetEmployeeName(name as STRING)
PRIVATE
  DECLARE ClearHistory( )
  'members
PROTECTED
  int m_employeeID
  float m_payrate
  float m_lastpay
  string m_name
  pointer m_history
END CLASS

CLASS manager, employee
'Constructor
  DECLARE VIRTUAL CalculatePay(hours as float)
  DECLARE manager( )
  DECLARE SetDividend(amount as float)
  DECLARE SetBonus(amount as float)
PROTECTED
  float m_dividend
  float m_bonus
END CLASS

SUB manager::CalculatePay(hours as float)
  m_lastpay = (m_payrate * hours) + m_bonus + (m_dividend * company_profit)
END SUB
```

The method CalculatePay is said to be overridden in the manager class. It does not replace the method since both methods do exist and are usable. What it does is allow for polymorphism when

using dynamically created objects. Polymorphism, as discussed previously, allows the compiler to call the correct method when your code doesn't know what kind of class a pointer points to. Let's say we have a subroutine that takes a pointer to an employee object.

```

POINTER man, emp
SETTYPE man, manager
SETTYPE emp, employee
man = NEW(manager,1)
emp = NEW(employee,1)
'fill in employee details here
'...
AddToPayroll(man)
AddToPayroll(emp)
DELETE man
DELETE emp

SUB AddToPayroll(emp as POINTER, hours as FLOAT)
    total_payroll += #<employee>emp.CalculatePay(hours)
ENDSUB

```

When the compiler encounters the `#<employee>emp.CalculatePay(hours)` statement it will call the correct method depending on the type of the class. For the manager object it will call the `CalculatePay` method of the manager class which adds in a bonus and dividend to the base wage. For the employee object it will call the `CalculatePay` method of the employee class which returns a straight wage * hours worked calculation.

The compiler accomplished this magic feat by using a virtual function table (VTABLE) which is just an array of addresses. Each array element contains the address of a method that was marked as VIRTUAL. When a class overrides a method from a base class that address is replaced with the address of the method declared in the derived class.

Scope Resolution

When writing a member subroutine it is sometimes necessary to call the base class version of a method, or to call a program subroutine that may share the same name as a classes method. For example the `CalculatePay` method of the manager class can be simplified by calling the `CalculatePay` method of the employee class:

```

SUB manager::CalculatePay(hours as float)
    m_lastpay = employee::CalculatePay(hours) + m_bonus + (m_dividend * company_profit)
END SUB

```

The double colon is used as the scope resolution operator in IW BASIC. When used to call a base class method the name of the base class is specified on the left hand side of the double colon. To execute a program subroutine, or API function, that might have the same name as a method simply prefix the call with the double colon. For example if you had a class called *graphics* that encapsulated some of the drawing commands used by IW BASIC and wanted to have a method called `LINE` which uses the built in `LINE` command you would need to tell the compiler to call the built in version.

```

SUB graphics::LINE(x as int, y as int, x2 as int, y2 as int)
  FRONTPEN m_window, m_fcolor
  'call the built in LINE command
  ::LINE(m_window,x,y,x2,y2)
END SUB

```

The THIS pointer

Every class method has a hidden first parameter known as the THIS pointer. It is a pointer to the instance of the created object. The THIS pointer is how the compiler knows which object a method subroutine is currently dealing with. Consider a method from the employee class:

```

SUB employee::SetEmployeeID(id as int)
  m_employeeID = id
END SUB

```

Internally the compiler sees it as:

```

SUB employee::SetEmployeeID(id as int)
  *<employee>this.m_employeeID = id
END SUB

```

Which are both functionally equivalent. You can use the THIS pointer in your own code to pass the current object to some other method or subroutine.

Imported methods

Methods can be imported from DLL's:

classdef.inc

```

class CExporter
  declare import CExporter()
  declare import _CExporter()
  declare import TestMethod()
endclass

```

DllMain.iwb (dll code, compile and create import library)

```

$include "classdef.inc"
export CExporter@CExporter
export CExporter@_CExporter
export CExporter@TestMethod

sub CExporter::CExporter()
  print "CExporter constructor called"
endsub

sub CExporter::_CExporter()
  print "CExporter destructor called"
endsub

sub CExporter::TestMethod()
  print "CExporter TestMethod called"
endsub

```

WinMain.iwb (exe code)

```

$include "classdef.inc"
$use "DllMain.lib"

```

```
CExporter c
c.TestMethod()
```

Design considerations

When designing a class that will be used in multiple source files place the class definitions in an include file and the method implementations in a source file that is part of the project. There is no need to use GLOBAL or EXTERN when referring to class methods as the compiler automatically handles this. Just \$include the file with the class definitions and add the methods source file to the project.

8.16 Inline Assembly

The compiler supports inserting assembly language source code directly into your IW BASIC source files. It is not the purpose of this text to instruct on how to program in assembly language, only on how to insert assembly language statements into your source code and accessing variables through assembly. Refer to the *Assembler documentation* accessible from the *Help* menu for more details on assembly syntax.

To begin an inline assembly block use the `_asm` keyword and end the block with the `_endasm` keyword. Any text between the two statements will be ignored by the compiler and is copied verbatim to the resultant compiler .a output file at whatever point the `_asm` statement appears. The statements must be in lower case. Example

```
DEF myInt as INT
_asm
    mov eax, 1
    add eax, 20
    mov dword [$myInt], eax
_endasm
PRINT myInt
```

The editor does not understand assembly code and will sometimes colorize assembler keywords. This is normal and will not affect the operation of the compiler.

Referring to global variables

Global variables, defined outside of any subroutine, can be referenced by your inline assembly code by using the \$ symbol followed by the variable name. The example above stores the number 21 in the myInt variable. The assembler itself is case sensitive and any reference to a global variable must match the case of the DEF/DIM statement.

All global variables are references to a memory location. The [] symbols is the same as a pointer dereferencing operation in IW BASIC.

Referring to local variables

Local variables are either defined in a subroutine or are one of the subroutines parameters. You can access a local variable by name or by computing its *stack offset*. The *stack offset* method is retained for backward compatibility.

When an IW BASIC subroutine is entered the register EBP contains a pointer to the current stack frame. All variables are accessed by using offsets to the EBX register depending on their type. For a subroutines parameters the first argument will be located at EBP+8. For variables defined in the subroutine the first variable is located at EBP-4. Consider this short example

```
SUB asmtest(b as INT),INT
DEF c as int
_asm
  lea esi,[ebp-4]      ;address of c
  mov  eax,[ebp+8]    ;value of b
  mov  [esi],eax      ;c = b
  add  [esi],dword 2  ;c=c+2
_endasm
RETURN c
ENDSUB
```

The parameter 'b' is located at ebp+8 and the variable 'c' is located at ebp-4. If the parameter is a reference such as a string, array or pointer then only an address to the reference is contained on the stack. You must use LEA to obtain that address and dereference as you would with any other pointer

```
SUB asmtest2(b as INT BYREF),INT
DEF c as int
_asm
  lea esi,[ebp-4]      ;address of c
  lea eax,[ebp+8]    ;address of variable pushed as b
  mov  eax,[eax]      ;dereference the address
  mov  [esi],eax      ;c = b
  add  [esi],dword 2  ;c=c+2
_endasm
RETURN c
ENDSUB
```

The other means of accessing local variables is enabled with either:
the /P option, when compiling from the command line; or
with the following code:

```
$option "/p 1"
```

By enabling this option you can change the above into this:

```
$option "/p 1" 'turn the option on
SUB asmtest(b as INT),INT
DEF c as int
_asm
  lea esi,[c] ;address of c
  mov  eax,[b] ;value of b
  mov  [esi],eax ;c = b
```

```

    add [esi],dword 2 ;c=c+2
    _endasm
RETURN c
ENDSUB
$option "/p 0" 'turn the option off

```

Assembly Data Segment

Raw data for IWBasic variables can be stored in an assembly data segment. In order to do so the variable has to be DECLARE'd and not DEF'd as regular variables are. The following demonstrates how this is accomplished:

```

declare rectangles as WINRECT

print rectangles.left, rectangles.top, rectangles.right, rectangles.bottom
print rectangles[1].left

_asm
segment .data
rectangles: dd 1,2,10,11 ; rectangles[0]
             dd 3,4,30,31 ; rectangles[1]
segment .text
_endasm

```

Another example with various types of data:

```

type mytype
  int a
  int b
endtype

typedef DWORD uint
declare iVal as int
declare sVal as string
declare tVal as mytype ' TYPE/ENDTYPE
declare dVal as DWORD ' typedef

print iVal, iVal[1]
print sVal
print tVal.a, tVal.b
print tVal[1].a, tVal[1].b
print dVal

_asm
segment .data
iVal: dd 6, 5, 4, 3, 2, 1
sVal: db "hello", 0
tVal: dd 100,101, 200,201
dVal: dd 500
segment .text
_endasm

```

Labels in assembly code

You can use labels for branching in your assembly code provided they do not interfere with any IWBasic source code label or subroutine name.

```
_asm
    mov eax, dword [$test]
    cmp eax, 0
    jz out1
    inc eax
    mov dword [$test], eax
out1:
_endasm
```

Register usage and context

You are free to use any register in your inline assembly. If creating straight assembly language subroutines you must preserve the contents of EBX, EDI and ESI by pushing them to the stack and popping them before your subroutine returns. If your assembly language subroutine uses any local stack space you must set up a proper stack frame with the EBP register and restore it before the subroutine returns.

8.17 Compiler Options

IWBasic's compiler has numerous options that may be invoked by the User. For backward compatibility all options are initially disabled/unused. Any given option can be enabled using one (and possibly more) of the three methods available listed below.

Global Options

The *Compiler Properties* dialog a multi-page dialog for entering global options. The [How-To»Set Compiler Preferences](#) section contains a complete discussion of the options available.

\$Options keyword

Most options may be controlled directly from inside the source code. Any changes here affect only the source file which contains them and do not change the entries stored in the IWBasic.ini file. However, changes made via this method will override the corresponding IWBasic.ini file entries, but just for the source file that contains the changes. The details of which options are available via this method are discussed [here](#).

Command Line

Some options may be controlled using command line switches when the compiler is invoked. As with the \$Options keyword method, the command line switches do not write to the IWBasic.ini file. The IWBasic.ini file is read and any switches will over ride the corresponding IWBasic.ini file entry for that compilation. The details of which option switches are available via this method are discussed [here](#).

8.17.1 Global Options

Global Options can be set with the *Compiler Preferences* dialog described in the [How-To»Set Compiler Preferences](#) section. Global option settings are stored in the IWBasic.ini file.

Global option entries can be overridden at compile time via [command line switches](#) and/or [\\$Option keyword](#) entries.

8.17.2 \$Option keyword

Compiler options can be set directly from the source code. Options set in this manner will override options set with the *Compiler Preferences* dialog described in the [How-To»Set Compiler Preferences](#) section. Any changes made in this manner will affect only the source file which contains them and do not change the global entries stored in the IWBasic.ini file.

The syntax for using the \$Options keyword is:

```
$option "name{=value}"
$option "name{ value}"
```

None of the syntax components are case sensitive. The following table is a summary of the available compiler options. Following the table are specific details for each of the options.

Option	Description
Float	Treats all numeric constants containing a decimal point as type FLOAT
Double	Treats all numeric constants containing a decimal point as type DOUBLE (default)
Codepage - / c	Sets the codepage for unicode strings.
ErrorLimit - / e	Sets the upper limit for compiler error messages.
IncludePath - / i	Add a search path for include files.
AsmVariable s - /p	Define formal and local variables for inline assembly code.
Warning - /w	Sets a new or modifies the current filter for warning messages.
Return	Force the compiler to not alter eax,edx when returning from a subroutine. Disables automatic "return 0" generation.
Intrinsic	Enable/disable intrinsic functions
Optimization	Enable/disable code generation optimizations.
CaseSensitiv e	Enable/disable case sensitive compilation
NoVtable	Enable/disable the virtual keyword for all classes.

OldLibsLocation	Controls where the compiler should create \$\$\$lib.link file. Previously the /bin directory was used, and the new IWBASIC will use project directory.
Strict	Show any possible problems, for example unknown type in DECLARE (even unused).
ZeroVariables	If active, local variables will be initialized to zero.
/m	Enable or disable \$main
/a	Compile and assemble
LibPath	Add a search path for .lib files.
QuoteLibPaths	Quote spaces in \$\$\$lib.link file.
ResolveLibs	Convert all relative paths passed to \$USE, to full paths.
DimOrder	Change the order of variables definition

Note: Numbers passed to \$option can be formatted in decimal, or hexadecimal with 0x prefix.

```
$option "codepage 65001"
$option "codepage 0xFDE9"
```

Float

value: ---

default state: OFF

Treats all numeric constants containing a decimal point as type FLOAT.

```
$option "float"
```

Note: The Float and Double options are mutually exclusive. Turning one on turns the other off.

Double

value: ---

default state: ON

Treats all numeric constants containing a decimal point as type DOUBLE.

```
$option "double"
```

Note: The Float and Double options are mutually exclusive. Turning one on turns the other off.

Codepage - /c

value: push, pop, utf8, ansi, or a number

default state: ANSI / CP_ACP / 0

```
$option "/codepage 65001" ' use a space or '=' as the separator
$option "/c push" ' save current codepage
$option "/c pop" ' restore saved codepage
$option "/c pop" ' restore
```

ErrorLimit - /e

value: push, pop, number

default state: 5

```
$option "ErrorLimit 10" ' show up to 10 errors
```

IncludePath - /i

value: a path, full or relative (to current file)

default state: EMPTY / environment variable INCLUDE

```
'$option "/i c:\downloads\programming\libs\sqlite" ' no c-style escape sequences
'$option "/i sqlite" ' subdirectory sqlite relative to current file
'$option "/i \..\sqlite" ' CurrentFileDirectory\..\sqlite
'$option "/i \..\sqlite" ' CurrentFileDirectory\..\sqlite
```

TIP: you can specify additional paths in environment variable INCLUDE: path1;path2;path3

TIP: you can specify additional paths in REG_MULTI_SZ "Include Paths" in

HKEY_CURRENT_USER\Software\IonicWind\IW BASIC

TIP: you can specify additional paths in the command line: /Ipath1 /Ipath2 "/Ilong path 3"

AsmVariables - /p

value: push, pop, 0,1,yes,no,on,off,enable,disable

default state: 0

```
$option "/p" ' enable
$option "/p 1" ' enable
$option "/p 0" ' disable
```

```
$option "/p 1"
sub mysub3(int t)
    int yy
    static int zz
    $emit mov eax,[t]
    $emit mov [yy],eax
    $emit mov [mysub3.zz],eax
endsub
$option "/p 0" ' optional: turn it off
```

Warning - /w

value: push, pop, enable, disable, default or a number

default state: 0

```
$option "/w push" ' save current filter
$option "/w pop" ' restore saved filter
$option "/w 0xFFFFFFFF" ' set all: enable all warnings
$option "/w 1" ' set all: disable all, enable warning 1
$option "/w disable:1" ' selective: disable warning 1
$option "/w enable:1" ' selective: enable warning 1
$option "/w default" ' set default warnings filter
```

currently implemented warning filters:

0x0001 uninitialized variables

0x0002 unreferenced variables

0x0004 missing return value

0x0008 size of local variables exceeds 32KB
 0x0010 variable/constant name collision
 0x0020 undeclared function
 0x0040 temporary string/udt assigned to pointer

```
' demo for uninitialized variable (do not execute this function)
$option "warning push" ' optional: save current mask
$option "warning disable:0x0001"
sub mysub(int pValue)
    int t
    mysub(t)
endsub
$option "warning pop" ' optional: restore saved mask

sub mysub2(int pValue)
    int t
    mysub2(t) ' Warning: Uninitialized variable: t
endsub
```

Return

value: push,pop,0,1,yes,no,on,off,enable,disable

default state: OFF

This option is used to return a value directly in eax register. If you turn this option ON, the compiler will not warn about missing return value, and will not return zero, as it does by default. Therefore, be sure to return zero from window handlers.

```
sub testsub(),int
    $emit mov eax,7
    $option "return disable"
endsub
$option "return enable"
print testsub()
```

Intrinsic

value: push, pop, 0,1,yes,no,on,off,enable,disable

default state: OFF

```
$option "intrinsic" ' same as $option "intrinsic enable"
int a=SIN(5)
```

Optimization .

value: push, pop, 0,1,yes,no,on,off,enable,disable

default state: OFF

Enables optimizing code generation

```
$option "optimization" ' same as $option "optimization enable"
```

Note: There is another optimization that can be invoked from the main menu *Projects/Options* dialog by entering **O1** in the *Advanced/Assembler Options* edit field. With this option active, the operand size specifier in assembly instructions like "mov reg,value" or push value will be removed..

CaseSensitive

value: push,pop,0,1,yes,no,on,off,enable,disable

default state: OFF

```
$option "CaseSensitive"
typedef XX int
typedef Xx word
typedef xX char
typedef xx int64
```

While this option is active, `__CASESENSITIVE__` condition will be defined. Use `$ifdef` to detect it:

```
typedef sbyte schar
#ifdef __CASESENSITIVE__
    typedef BYTE char
#endif
```

Case sensitive compilation mode does not affect built-in variable types, so `char` and `CHAR` are the same type.

Case sensitivity allows faster compilation because the compiler doesn't have to alter the case of everything in order to make comparisons.

If case sensitive compilation is enabled, anything but build-in base types (and the list below) will be case sensitive - functions, constants, id's, variables, structures, typedefs.

The following will be case insensitive:

`def`, `dim`, `$error`, `$warning`, `typedef`, `$typedef`, `$undef`, `gosub`, `goto`, `jump`, `label`, etc... and includes all the keywords defined as `{command}` in the `.inc` files.

The compiler will search in all possible locations (using case insensitive method) to give you a hint, with the proper case: "did you meant TheSymbol?"

```
sub b()
    point p
endsub
```

Error message: *File: x.iwb (19) unknown type point, did you meant POINT ?*

NoVtable

value: push,pop,0,1,yes,no,on,off,enable,disable

default state: OFF

Activating this option disables the virtual keyword for all classes defined afterward. This in turns moves all class variables 4 bytes "up"; the first variable will be located at the `THIS` pointer (like in structures).

```
$option "novtable"
class CRect
    int left
    int top
    int width
    int height
    declare Set(int l, int t, int w, int h)
'
    declare virtual !!! not possible !!!
endclass
```

This class behaves like a structure, in the same way as it would in C++: 'left' member is at offset 0, 'top' at 4 ...

Another example program:

```
class CWinRect
int left
int top
int right
int bottom
endclass

CWinRect c

c.left = 1
c.top = 2
c.right = 3
c.bottom = 4

pointer p = &c
print "values:", *<int>p[0], *<int>p[1], *<int>p[2], *<int>p[3]
print "offsets: ", &c.left-&c, &c.top-&c, &c.right-&c, &c.bottom-&c
```

Output:

```
values:4206666 1 2 3
offsets: 4 8 12 16
```

With novtable enabled:

```
values: 1 2 3 4
offsets: 0 4 8 12
```

OldLibsLocation

value: push,pop,0,1,yes,no,on,off,enable,disable

default state: OFF

```
$option "oldlibslocation on"
```

Controls where the compiler should create \$\$\$lib.link file. When ON, the /bin directory is used, and when OFF, the current project's directory is used..

Strict

value: push,pop,0,1,yes,no,on,off,enable,disable

default state: OFF

```
$option "strict on"
```

Show any possible problems, for example unknown type in DECLARE (even unused).

ZeroVariables

value: push,pop,0,1,yes,no,on,off,enable,disable

default state: OFF

If active, local variables will be initialized to zero.

```
$option "zerovvariables on"
sub ZeroLocalVariables()
  int a ' will be zero
  int b ' will be zero
  int c ' will be zero
```

```

    print a,b,c ' outputs "0 0 0"
endsub
ZeroLocalVariables()

```

/m.

value: push,pop,0,1,yes,no,on,off,enable,disable

default state: OFF

Used to enable or disable \$main

Used as /m in the command line when compiling a single file to exe/dll

If you use \$option "/m" instead of \$MAIN, your program will start "at the top of source file", from the first expression.

```

$option "/m"           ' use $main
$option "/m 1"         ' use $main
$option "/m yes"       ' use $main
$option "/m on"        ' use $main
$option "/m enable"    ' use $main
$option "/m 0"         ' cancel $main
$option "/m no"        ' cancel $main
$option "/m off"       ' cancel $main
$option "/m disable"   ' cancel $main

```

/a - push,pop,0,1,yes,no,on,off,enable,disable. .

value: push,pop,0,1,yes,no,on,off,enable,disable

default state: OFF

When disabled the compiler will generate an assembly source file; otherwise an object file will be created.

```

$option "/a" ' same as $option "/a on"

```

NOTE: If the output file passed to the parser has an .o or .obj extension, an object file will be generated using the local nasmw.exe.

LibPath

value: directory

default state: NULL

Add a search path for .lib files. Used with ResolveLibs option to write full lib paths to \$\$\$lib.link file.

```

$option "LibPath c:\downloads\programming\libs\MyDir" ' no c-style escape sequences
$option "LibPath MyDir"           ' subdirectory MyDir relative to current file
$option "LibPath ..\MyDir"        ' CurrentFileDirectory\..\MyDir
$option "LibPath ..\MyDir"        ' CurrentFileDirectory\..\MyDir

```

QuoteLibPaths

value: push,pop,0,1,yes,no,on,off,enable,disable

default state: off

Quote spaces in \$\$\$lib.link file. Useful option if you want to use another linker.

ResolveLibs

value: push,pop,0,1,yes,no,on,off,enable,disable

default state: off

Convert all relative paths passed to \$USE, to full paths.

DimOrder

value: push,pop,left,right,default

default state: default

Change the order of variables definition.

```
int a,b,c,d ' the default is 'right', the 'd' variable is defined first.
```

8.17.3 Command line

Compiler options can be set directly from the command line. Options set in this manner will override options set with the *Compiler Preferences* dialog described in the [How-To»Set Compiler Preferences](#) section but do not change the global entries stored in the IWBASIC.ini file..

The proper syntax for options is:

/[option name][parameter]

without a space between option name and parameter. The option name is not case sensitive.

The following table describes available options.

Option	Description
/As	Compile and assemble. Optional <i>s</i> is passed to nasm.
/Bpath	Include a file <path> before parsing the main file. See Note below.
/Cnumber	Changes the default codepage from Ansi (CP_ACP) to <number>. This option ensures that unicode strings (L"") are properly converted from multibyte character set (controlled by the User's text editor) to unicode binary data. If the User's editor is working with utf-8, set this option to /C65001 (CP_UTF8). The User will be warned if a codepage is not valid for the current system.
/Dname {=value}	Define a numeric constant.
/Enumber	Show up to <number> errors; default is 5. Does not apply to syntax errors.

/G	Resolve relative library paths.
/H or /?	Displays the help information shown in Fig. 1, below.
/Ipath	Adds a search path for include files. Pass full path, or a relative path to current file (\..\ is supported). See Note below.
/Lname.dll	Loads a plugin (Not currently available). See Note below.
/M	For single file compilation, use \$MAIN.
/O	Show only filename in warnings/errors.
/P	<p>Defines arguments and local variables for use in inline assembly. Take care that variable names don't interfere with assembler registers or instructions.</p> <pre> sub sum(int a, int b),int int c _asm mov eax,[a] add eax,[b] mov [c],eax _endasm return c endsub </pre>
/Rname	<name> specifies a path to response file, where each line is a pathname to an IWBasic source file. See Note below.
/S	Be strict.
/Wnumber	<p>Disable warnings. If not specified then all warnings are enabled. Currently implemented warning filters:</p> <ul style="list-style-type: none"> 0001 (C0001) uninitialized variables 0002 (C0002) unreferenced local variables 0004 (C0003) missing return value 0008 (C0004) size of local variables exceeded 32KB 0010 (C0005) variable/constant name collision 0020 (C0006) undeclared function 0040 (C0007) temporary string/udt assigned to pointer

NOTE: In the command line no space is allowed for an option, only in path names (the whole option must be quoted):

i.e. /Ishortpath "/Ilong path".

When passing "folders" in the command line, they should not end with a backslash. If there must be one at the end, and the path is quoted, the User should use two backslashes:

"/SOMEOPTIONc:\blah\"

```
Usage: IWBPARSE [options] inputfile [options] [outputfile] [options]
Available options:
/As          Compile and assemble. Optional s is passed to nasm.
             If s is not specified, '-f win32' will be used.
/Bpath       Include file <path> before parsing the main file.
/Cnumber     Change default codepage for unicode strings. For utf-8 use /C65001.
/Dname{=value} Define a condition or constant.
/En          Show up to n-errors (default 5).
/G           Resolve relative lib paths.
/H or /?    Show this help.
/Ipath       Add a search path for include files.
/M           For single file compilation, use $main.
/O           Show only file name in warnings/errors.
/P           Define arguments and local variables for inline assembly
/R           Inputfile specifies a path to response file, where
             each line is a pathname to an IWBasic source file.
/S           Be strict
/Wnumber     Disable warnings:
             1 uninitialized variables
             2 unreferenced variables
             4 missing return value
             8 size of local variables exceeds 32KB
             16 variable/constant name collision
             32 structure redefinition
             64 temporary string/udt assigned to pointer
             128 missing base interface
```

1. Command Line - Help Display

8.18 Exception Handling

Exceptions are events that normally terminate program execution when they occur. They can be either hardware-based (like out-of-memory) or software-based (like division by zero). In either case what often results is a program "crash". It would be nice to be able to prevent all crashes. That can't be done. So the next best thing is to prevent the crashes that can be prevented and learn the most we can from those that can't be recovered from. For the latter we'd like to display error information and trap the internal state of the application to help diagnose the problem. This is especially helpful with intermittent problems that cannot be reproduced easily.

The method that IWBASIC uses to "handle" these exceptions is called structured exception handling.

The first step in implementing exception handling is to visually identify portions of the code that could possibly generate an exception. Good examples are places where a division is taking place or the SQRT function is being used.

NOTE: Throughout the rest of this section numerous examples show a typical "divide by zero" exception and the responses to it. This does not imply that that particular exception is the only exception that can be generated and needs to be addressed. "Divide by zero" exceptions are simply easy to create and understand for demonstration purposes.

The following demonstrates code that will cause an exception:

```
int a = 1, b = 2
int y = 12 / (a - 1) ' division by zero
int z = SQRT(a - b) ' square root of a negative number
```

As an example, let's examine the following program:

```
openconsole
int a = 0
'int a = 1
print 1/a
print "success"

print "Any key to end"
waitcon
closeconsole
end
```

When the program is compiled and run, nothing is printed. That's because the divide by zero has created an exception and the program has crashed.

Once the potentially problem code is identified it has to be marked in some way for the compiler to recognize it needs to be dealt with. This is accomplished by bracketing the code with the **TRY** / **ENDTRY** keywords.

```
openconsole
int a = 0
'int a = 1
TRY
  print 1/a
ENDTRY
print "success"

print "Any key to end"
waitcon
closeconsole
end
```

So now the code is set up to recognize an exception. If $a \neq 0$ when the TRY was entered the division would be done and the result printed just as though the TRY/ENDTRY keywords weren't there. The other two print statements would also be printed.

If the code is compiled and run as shown the divide by zero error will be trapped and only "success" and "Any key to end" will be printed. That indicates the program ran to completion but does not tell us why no number was printed and lies in telling us it was a success.

Let's move the "print 'success' " line to inside the TRY/ENDTRY block:

```
openconsole
int a = 0
'int a = 1
TRY
  print 1/a
  print "success"
ENDTRY

print "Any key to end"
waitcon
```

```
closeconsole
end
```

If the above code is compiled and run with $a = 1$ the results will be as expected, all three print statements will execute.

With $a = 0$ only the "print 'Any...'" statement is executed.

Within a TRY/ENDTRY block, when an exception occurs any statements following the line that causes the exception are skipped. Conversely, when no exception is encountered in the block the code executes normally.

NOTE: Again, it must be stressed that within a given TRY/ENDTRY block there may be numerous exceptions generated for a multitude of different reasons. The examples in this section are assuming that the only exception that can be generated is the one being caused by the "print 1/a" statement.

Up to this point the missing print of the division when $a = 0$ has not been addressed. The following code shows one method of addressing the issue:

```
openconsole
int a = 0
'int a = 1
WHILE 1
  TRY
    print 1/a
    print "success"
    BREAK
  ENDTRY
  print "Divide by Zero Error- Using default"
  a=1
ENDWHILE

print "Any key to end"
waitcon
closeconsole
end
```

If $a \neq 0$ the division result is printed; "success" is printed; and the BREAK statement is executed to exit the WHILE/ENDWHILE loop. If $a = 0$ the division generates an exception; nothing is printed; the TRY/ENDTRY block is exited; the "Divide error..." line is printed; a is set to a default non zero value; the program is returned to the WHILE statement; and the division doesn't generate an exception this time through.

There is another way to accomplish the preceding. Observe the following code:

```
label_a:
  TRY
    print 1/a
    print "success"
  ENDTRY
  CATCH
    print "Divide by Zero Error- Using default"
    a=1
    goto label_a
  ENDCATCH
```

First notice the **CATCH / ENDCATCH** keywords. These denote a block of code that gets

executed only if an exception is generated. If the optional CATCH block is used, there can be one CATCH block for each TRY block or there can be multiple TRY blocks using a single CATCH block in a single application.

An exception inside a TRY block sets an internal flag to TRUE (initialized to FALSE when entering a TRY block), and continues program execution outside the current TRY block. That internal flag is examined each time a CATCH block is encountered. If the flag is TRUE the CATCH block code will be executed; otherwise it will be skipped.

CATCH blocks can be anywhere, even in a separate subroutine or DLL. However, a CATCH block's code only works in the 'current' thread. CATCH code must be executed in the same thread as the recent TRY, otherwise it will return exception information for another thread.

The following shows three TRY/CATCH pairs with recovery for the first two and the third treated as a non-recoverable exception.

```
openconsole
int a,b,c
exec_a:
  try
    print 1/a
  endtry
  catch
    a=1
    goto exec_a
  endcatch

exec_b:
  try
    print 1/b
  endtry
  catch
    b=-1
    goto exec_b
  endcatch

exec_c:
  try
    print 1/c
  endtry
  catch
    print "fatal error"
  end
  endcatch

print "Any key to end"
waitcon
end
```

This example demonstrates the nesting of TRY blocks and the use of multiple TRY blocks with one CATCH block:

```
openconsole
int a,b,c

exec_a:
  try
```

```

    print 1/a
exec_b:
  try
    print 1/b
    try
      print 1/c
    endtry
  endtry
endtry

catch ' exception in one of three divisions
  if a=0
    a=1
    print "problem with a"
    goto exec_a
  elseif b=0
    ' 'a' succeeded
    print "problem with b"
    b=-1
    try      'needed to account for the fact we are not going to the
      endtry 'top level of the nesting. This will be repeated for each level skipped
      goto exec_b
    else
      ' c must be zero
      print " fatal problem with c"
      end
    endif
  endcatch

print "Any key to end"
waitcon
end

```

The number of nesting levels of TRY blocks is unlimited.

NOTE: Due to having to insert extra TRY blocks inside the CATCH when using GOTO in nested TRY blocks it is highly recommended that GOTO not be used in nested TRY blocks.

Up to this point nothing has been discussed about how to collect and display information concerning an exception.

IW BASIC provides two functions for use in a CATCH block to supply that information.

1. [GetExceptionCode](#) - returns the code (a 32-bit integer) of the exception.
2. [GetExceptionInformation](#) - returns a pointer to a structure containing additional information about the exception. Through this pointer the machine state that existed at the time of a hardware exception can be accessed..

The following is an example of their use:

```

#include "windowssdk.inc" ' needed for definition of EXCEPTION_POINTERS and EXCEPTION_RECORD
int a=0
try
  print "dividing by zero"
  print 1/a
  print "divide ok"
endtry

catch
  print "exception code: 0x", hex$(GetExceptionCode())

```

```
pointer pep = GetExceptionInformation()
settype pep, EXCEPTION_POINTERS

print "exception address: 0x", hex$(*pep.*<EXCEPTION_RECORD>ExceptionRecord.ExceptionAddress)
endcatch
print "finished"
```

Output:

```
dividing by zero
exception code : 0xC0000094
exception address: 0x40141E
finished
```

The following example uses what has been covered so far with an additional subroutine and some special IW BASIC constants.

```
uint code = 0
int a = 0
'int a = 1
  try
    print 1/a
    ' other code here
    print "success"
  endtry

if (WasThereAnException(&code)) then showerror(__FILE__, __LINE__, code)

'dummy()
end

sub dummy
  catch
    print "in dummy sub"
  endcatch
endsub

sub WasThereAnException(opt pointer pdwCode), int
  int t = 0
  catch
    t = 1
    if (pdwCode) then *<int>pdwCode = GetExceptionCode()
  endcatch
  return t
endsub

sub showerror(string file, uint line, uint code)
  MESSAGEBOX 0, USING("Error & in &#", HEX$(code), file, line), ""
endsub
```

First, let's examine what happens when $a = 1$.

The TRY block is entered and the exception flag is set to FALSE. There is no exception so "success" is printed.

The WasThereAnException sub is called in the IF statement.

Inside the sub the CATCH block is skipped since the internal exception flag is FALSE.

The sub returns 0 which causes the IF statement to not call the showerror sub.

Now, with $a = 0$.

The TRY block is entered and the exception flag is set to FALSE. There is an exception so "success" is not printed.

The exception flag is set to TRUE

The WasThereAnException sub is called in the IF statement.

Inside the sub the CATCH block is enter since the internal exception flag is TRUE..

The exception code is stored in the memory location for "code".

The sub returns 1 which causes the IF statement to call the showerror sub.

The showerror sub opens the message box and displays the current file name, the current line number, and the exception code.

Examine the code again and notice the "dummy" sub. The statement that calls the sub is commented out so it wouldn't execute. Let's remove the comment so the call to dummy will occur.

Again. evaluate with $a = 0$.

Everything happens exactly as described in the previous paragraph with $a = 0$. However, when the messagebox is closed and the "dummy" sub is executed, the "in dummy sub" line is printed. That is because the exception flag can only be reset upon entering a TRY block. Remember in the early examples that the looping was reentering the TRY block which in turn was resetting the exception flag. The resulting action in this scenario demonstrates the need for extreme caution when using TRY/CATCH blocks.

There is another way of creating exceptions that hasn't been touched on. There are circumstances where it would useful to be able to use the TRY/CATCH mechanism without having to have an actual hardware or software failure.

The THROW keyword can be used for this purpose. When used it creates an exception with a code of 0. It must be used with an argument (a pointer, number, string) that identifies the specific exception.

The following demonstrates the use of the THROW keyword:

```
$include "windowssdk.inc"
#define DEMO 2      '<===== change to 0, 1, 2 to see different results

    try
        int a=7
    $if DEMO==0
        throw a ' throw VARIABLE INT
    $elif DEMO==1
        throw 1234 ' throw NUMBER
    $else
        throw "help" ' throw STRING
    $endif
    endtry

    catch
        pointer excptPtrs = GetExceptionInformation()
        settype excptPtrs, EXCEPTION_POINTERS
        settype *excptPtrs.ExceptionRecord, EXCEPTION_RECORD
    $if DEMO==0
        ' throw VARIABLE INT
        pointer pValue = pointer(*excptPtrs.*ExceptionRecord.ExceptionInformation[0])
```

```

    print "throw value: ", *<int>pValue
$elif DEMO==1
    ' throw NUMBER
    print "throw value: ", *excptPtrs.*ExceptionRecord.ExceptionInformation[0]
$else
    ' throw STRING
    pointer pszValue = pointer(*excptPtrs.*ExceptionRecord.ExceptionInformation[0])
    print "throw value: ", *<string>pszValue
$endif
endcatch

```

THROW can be used anywhere within a program. If it is use outside of a TRY block it will be "caught" by the debugger as a second chance exception, if in DEBUG mode. If THROW is not used in a TRY and not in DEBUG mode the program will crash.

The last keyword that requires mention is LEAVE. In looping blocks of code (FOR/NEXT, WHILE/ENDWHILE, DO/UNTIL) it is sometimes desirable / necessary to exit the block before all the iterations have been completed.

BREAK / BREAKFOR are used for that purpose. In a TRY block, LEAVE is used in exactly the same way, to do exactly the same thing.

8.18.1 Exception Codes

Exception Codes

[Top](#) [Previous](#) [Next](#)

The following table identifies the exception codes that can occur due to common programming errors. These values are defined in Winbase.h and Winnt.h.

Return code	Description
EXCEPTION_ACCESS_VIOLATION 0xC0000005	The thread attempts to read from or write to a virtual address for which it does not have access. .
EXCEPTION_ARRAY_BOUNDS_EXCEEDED 0xC000008C	The thread attempts to access an array element that is out of bounds, and the underlying hardware supports bounds checking.
EXCEPTION_BREAKPOINT 0x80000003	A breakpoint is encountered.
EXCEPTION_DATATYPE_MISALIGNMENT 0x80000002	The thread attempts to read or write data that is misaligned on hardware that does not provide alignment. For example, 16-bit values must be aligned on 2-byte boundaries, 32-bit

	values on 4-byte boundaries, and so on.
EXCEPTION_FLT_DENORMAL_OPERAND 0xC000008D	One of the operands in a floating point operation is denormal. A denormal value is one that is too small to represent as a standard floating point value.
EXCEPTION_FLT_DIVIDE_BY_ZERO 0xC000008E	The thread attempts to divide a floating point value by a floating point divisor of 0 (zero).
EXCEPTION_FLT_INEXACT_RESULT 0xC000008F	The result of a floating point operation cannot be represented exactly as a decimal fraction.
EXCEPTION_FLT_INVALID_OPERATION 0xC0000090	A floatin point exception that is not included in this list.
EXCEPTION_FLT_OVERFLOW 0xC0000091	The exponent of a floating point operation is greater than the magnitude allowed by the corresponding type.
EXCEPTION_FLT_STACK_CHECK 0xC0000092	The stack has overflowed or underflowed, because of a floating point operation.
EXCEPTION_FLT_UNDERFLOW 0xC0000093	The exponent of a floating point operation is less than the magnitude allowed by the corresponding type.
EXCEPTION_GUARD_PAGE 0x80000001	The thread accessed memory allocated with the PAGE_GUARD modifier.
EXCEPTION_ILLEGAL_INSTRUCTION 0xC000001D	The thread tries to execute an invalid instruction.
EXCEPTION_IN_PAGE_ERROR 0xC0000006	The thread tries to access a page that is not present, and the system is unable to load the page. For example, this exception might occur if a network connection is lost while running a program over a network.
EXCEPTION_INT_DIVIDE_BY_ZERO 0xC0000094	The thread attempts to divide an integer value by an integer divisor of 0 (zero).
EXCEPTION_INT_OVERFLOW 0xC0000095	The result of an integer operation causes a carry out of the most significant bit of the result..
EXCEPTION_INVALID_DISPOSITION 0xC0000026	An exception handler returns an invalid disposition to the exception dispatcher. Programmers using a high-level language such as C should never encounter this exception.
EXCEPTION_INVALID_HANDLE 0xC0000008	The thread used a handle to a kernel object that was invalid (probably because it had been closed.)
EXCEPTION_NONCONTINUABLE_EXCEPTION 0xC0000025	The thread attempts to continue execution after a non-continuable exception occurs.
EXCEPTION_PRIV_INSTRUCTION 0xC0000096	The thread attempts to execute an instruction with an operation that is not allowed in the current computer mode.
EXCEPTION_SINGLE_STEP 0x80000004	A trace trap or other single instruction mechanism signals that one instruction is executed.
EXCEPTION_STACK_OVERFLOW 0xC00000FD	The thread uses up its stack.
STATUS_UNWIND_CONSOLIDATE	A frame consolidation has been executed.

0x80000029

8.19 Threads

IWBASIC supports multithreaded processes via the windows api. Thread-private variables are supported via the THREAD keyword.

The **thread** keyword must be specified first, to take effect (first in a line). The compiler updates an internal UDT as it sees the thread variables. To share thread variables between multiple source files, define them in an include file, or use the same order in all source files.

```

declare import,CreateThread(pointer q,int q,int q, pointer q,int q,pointer q),int
declare import,WaitForSingleObject(int q,int q)
declare import,CloseHandle(int q)

' define some per-thread variables
thread int x,y
thread POINT pt

' main thread
x = 1
y = 2
pt.x = 3
pt.y = 4

print "main thread:"
DumpVariables(0) ' shows 1,2,3,4

' run a secondary thread

print "\nsecondary thread:" ' shows 0,0,0,0
int dwId
int hThread = CreateThread(0,0,&DumpVariables,0,0,&dwId)
if (hThread)
WaitForSingleObject(hThread, -1)
CloseHandle(hThread)
endif

sub DumpVariables(int reserved)
print "thread int x = ",x
print "thread int y = ",y
print "thread POINT pt.x = ", pt.x
print "thread POINT pt.y = ", pt.y
endsub

```

8.20 Macros

IWBASIC supports macros which are a set of instructions that are represented in an abbreviated format. The \$MACRO directive is used to define a macro. Once defined macros are called in basically the same way that functions are called.

Syntax

\$MACRO name(param1, param2, ...)(expression1 :expression2:...)

Parameters

name - A unique name for the macro. The name may be the same as a function name.

param1, ... - Symbols representing the variables being passed to the macro.

expression1,... - Expressions or functions to be performed. The param symbols are placed in the appropriate locations within the expressions.

```
$macro sum(a,b) (a+b)
$macro sum(a,b,c) (a+b+c)

print sum(1,2)
print sum(1,2,4)
'multi expression example
$macro INITPOINT(pt,xx,yy) (pt.x=xx : pt.y=yy)

POINT p
INITPOINT(p,3,4)
```

Note: The = operator inside a macro will always assign a value, and == will compare. This exception makes it possible to assign values in macros.

A macro will override a function with the same name and number of parameters:

```
declare import,sum(int a,int b),int
declare import,sum(int a,int b,int c),int
print sum(1,2) ' function

$macro sum(a,b) (a+b)
print sum(1,2) ' macro
print sum(1,2,3) ' function
```

A function can be called from inside a macro:

```
$macro ListboxDeleteAllItems(win,id) (SENDMESSAGE(win, LB_RESETCONTENT, 0, 0, id))

ListboxDeleteAllItems(d1, IDC_MYLISTBOX)
```

In macros defined with multiple expressions, only the first expression (leftmost) can be used to return a value. The return value from additional expressions will be ignored, and if necessary, temporary strings or UDT's will be deleted.

\$macro mymacro(a,b)(a:b)

```
print mymacro(APPEND("a","b"), APPEND("c","d"))
```

The first APPEND will be printed, the second APPEND will be executed and it's temporary result will be deleted before the macro returns.

The look-up precedence in macros is:

1. symbols from macro definition - \$macro name(*this, and _this*) (expression)
2. variables
3. constants

Therefore, if the symbol "A" is encountered in a macro expression, the compiler will first try to find macro parameter "A", then "A" variable, and finally a constant "A".

A macro can be undefined:

```
$UNDEF MacroName NumberOfParameters
```

```
$undef sum 2
$undef sum 3
```

To check if a macro is defined, use `$IFDEF MacroName`.

Macros are supported in enumerations and `$IF` conditions.

There is a special predefined macro: **defined(name)**

It returns TRUE if the parameter 'name' is defined. The 'defined' keyword is now reserved.

```
$define MYCOND
$define WIN32
'$define MAC

$if defined(WIN32) && defined(MAC)
  $error "WIN32 and MAC is defined"
$elif defined(WIN32) && defined(MYCOND)
  $error "WIN32 and MYCOND is defined"
$elif defined(WIN32) && !defined(MYCOND)
  $error "WIN32 is defined, but MYCOND is not defined"
$elif defined(MAC) && defined(MYCOND)
  $error "MAC and MYCOND is defined"
$elif defined(MAC) && !defined(MYCOND)
  $error "MAC is defined, but MYCOND is not defined"
$else
  $error "unhandled case"
$endif
```


General Programming

Part



9 General Programming

9.1 Text Only Programs

Console Window

The console window, also called a *text console*, is similar to a command prompt display. The console window can be used for text input and output only. Graphics are not supported for console mode programs.

To open a console window in your program use the [OPENCONSOLE](#) command. When you are done with the console window issue a [CLOSECONSOLE](#) command. OPENCONSOLE and CLOSECONSOLE are not needed if you compile for a *Console* target but it does not hurt to include them.

If compiling for a *Windows* target you must use OPENCONSOLE or the program will lock trying to output or input to a non-existent text console. Window only allows one console window per process.

Displaying Text

To display text in the console window use the PRINT statement. The print statement has the following syntax.

```
PRINT {window,}param1 {,param2...} {,}
```

The first optional parameter is to output text in a regular window and will be discussed in the section on using windows.

The optional trailing comma tells PRINT not to issue a carriage return at the end of the line. This allows using multiple PRINT statements to work with the same line of text.

A print statement with no parameters will move the console window cursor to the next line. The following program demonstrates a simple console program.

```
OPENCONSOLE
DEF name$,address$,city$,state$,zip$:STRING
DEF income:FLOAT
name$ = "John Doe"
address$ = "123 Anywhere St"
city$ = "Middle Town"
state$ = "WI"
zip$ = "55555"
income = 38.5
PRINT name$
PRINT address$
PRINT city$," ",state$," ",zip$
```

```
PRINT
PRINT "Median Income:",
PRINT income
PRINT
PRINT "Press Any Key To Close"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Changing Colors

To display your text in different colors in the console window use the COLOR statement. The COLOR statement has the syntax of:

COLOR foreground, background

Where background and foreground are positive numbers from 0 to 15. Refer to figure 16.1 for the colors supported by the console window.

In the previous example, insert the statement:

```
COLOR 14,1
```

Right before all the PRINT statements to see yellow text on a blue background.

Positioning Text

You can position your text anywhere in the console window by using the LOCATE statement. The LOCATE statement has the syntax of:

LOCATE y, x

where y is the vertical character position and x is the horizontal character position. The origin is the upper left corner at character position 1,1. This short example demonstrates the LOCATE statement:

```
OPENCONSOLE
LOCATE 10,1
PRINT "Position 10,1"
LOCATE 10,50
PRINT "Position 10,50"
LOCATE 1,1
PRINT "Position 1,1"
LOCATE 1,50
PRINT "Position 1,50"
LOCATE 12,1
PRINT "Press Any Key To Close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

figure 16.1

COLOR number	Color Produced
0	BLACK
1	BLUE
2	GREEN
3	CYAN
4	RED
5	MAGENTA
6	BROWN
7	WHITE
8	GRAY
9	LIGHT BLUE
10	LIGHT GREEN
11	LIGHT CYAN
12	LIGHT RED
13	LIGHT MAGENTA
14	YELLOW
15	HIGH INTENSITY WHITE

Clearing The Window

To clear the console window use the command [CLS](#). CLS removes all text the console window.

```

OPENCONSOLE
PRINT "CCCCCCCCCCCCCCCCCCCC"
PRINT "XXXXXXXXXXXXXXXXXXXX"
PRINT "Press any key to clear this window"
DO:UNTIL INKEY$ <> ""
CLS
LOCATE 12,1
PRINT"Now press any key to close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END

```

Waiting for a key

To pause your program and wait for a key to be pressed you can use the WAITCON command as an alternative to looping with INKEY\$

```

OPENCONSOLE
PRINT "Press any key to close"
WAITCON
CLOSECONSOLE
END

```

Getting Input

So far, we have concentrated on displaying and manipulating text in the console window. For a program to be interactive, we need some way to get data from the user into our program. IW BASIC provides one statement for the console window INPUT and one function INKEY\$.

The [INPUT](#) statement allows your program to receive information from the user directly into any assignable variable type. The syntax of the INPUT statement looks like this:

```
INPUT {?"prompt",} variable
```

The first optional parameter is a literal string used as the prompt for the input statement. The INPUT statement keeps collecting user key presses until the <ENTER> key is pressed at which time the input is stored in the variable.

```
OPENCONSOLE
CLS
DEF name$:STRING
INPUT "Enter your name:",name$
PRINT "Hello ",name$," Nice to meet you"
PRINT name$,"Please press any key to close!"
DO: UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The [INKEY\\$](#) function waits for the user of your program to press one key and returns the character pressed. This return value can be assigned to either a STRING or CHAR variable. The syntax of the INKEY\$ function is:

```
STRING|CHAR = INKEY$ { ( raw ) }
```

The optional raw parameter if equal to 1 tells INKEY\$ to return virtual key codes. It is important to note that before your program executes the INKEY\$ function that the user of your program may have typed a number of keys. Always check the return value for the expected results.

Example:

```
OPENCONSOLE
DEF key$:STRING
LOCATE 1,1
PRINT "INKEY$ demonstration. ",
PRINT "Press <ENTER> to end program"
PRINT "Press any other keys to display them"
DO
  key$ = INKEY$
  IF Key$ <> ""
    LOCATE 10,1
    COLOR 14,1
    PRINT "You Pressed >",
    COLOR 2,0
    PRINT key$,
  ENDF
UNTIL key$ = CHR$(13)
```

```
CLOSECONSOLE
END
```

9.2 File Operations

File operations allow you to read and write data to permanent storage devices such as a hard drive, floppy or removable media. They can also be used to send data to peripherals attached to your computer, such as a printer.

Opening files for reading and writing

The [OPENFILE](#) function initiates the connection between your program and the outside world. It has as syntax of:

Error = [OPENFILE](#) (Variable, filename, flags)

The *variable* must be either of type FILE for an ASCII file or BFILE to open the file in binary (raw) mode.

The *flags* parameter is a quoted string and can be one of "R" for reading, "W" for writing, "A" for appending to an existing file. For binary files the flags parameter can also be "R+" to open an existing file for both reading and writing.

The function will return 0 if the file was successfully opened or 1 if the file could not be created or opened. If the function fails you should not attempt to read or write to the file.

Appending opens the file, or creates it if it doesn't exist, and sets the file pointer to the end of the file (EOF). The *W* mode will create a new file, or empty an existing one, and set the file pointer to the beginning of the file. You are allowed to read from the file in both *W* and *A* modes.

```
DEF myfile as BFILE
IF OPENFILE(myfile, "C:\\temp\\temp.txt", "W") = 0
    WRITE myfile, 0xFFFF
    CLOSEFILE myfile
ENDIF
```

Closing open files

After you are finished reading or writing to a file you must close it to make sure that system resources are returned to Windows. If you leave too many files open at once you will receive an error message. Use the [CLOSEFILE](#) statement to close any open file.

Example:

```
OPENCONSOLE
DEF myfile:FILE
IF (OPENFILE(myfile, "C:\\IW BASIC TEST.TXT", "W") = 0)
    WRITE myfile, "This is a test"
    CLOSEFILE myfile
    PRINT "File created successfully"
ELSE
```

```
    PRINT "File could not be created"
ENDIF

PRINT "Press Any Key To Close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The previous example attempts to open a file for writing in the C:\ directory called "TWBASICTEST.TXT". After opening the file, it writes one line of text to the file and then closes the file.

Writing to the file

In the example a new function was introduced, [WRITE](#). WRITE returns 0 on success and can work with both binary and ASCII (text) files. WRITE expects two arguments, the variable initialized with OPENFILE and the data to be written.

How the data is written depends on whether or not an ASCII (FILE) variable or binary (BFILE) variable was used to open the file. In a binary file all data is written in raw form and will occupy the full size of the dimensioned variable. For example a standard STRING is 255 bytes:

```
DEF myfile as BFILE
DEF str as STRING
str = "hello"
IF OPENFILE(myfile, "C:\\temp\\temp.bin", "W") = 0
    WRITE myfile, str
    CLOSEFILE myfile
ENDIF
```

The length of the file will be 255 bytes in this case. However using an ASCII file the WRITE statement will only write the contents of the string and automatically terminate the string with a linefeed/carriage return pair.

```
DEF myfile as FILE
DEF str as STRING
str = "hello"
IF OPENFILE(myfile, "C:\\temp\\temp.txt", "W") = 0
    WRITE myfile, str
    CLOSEFILE myfile
ENDIF
```

The length of the file would be 7 bytes in this case and be loadable in any text editor.

Numeric data is also treated differently in ASCII and binary files. In binary mode numeric data is written in its raw form which occupies the same amount of bytes it would in memory. For example an INT variable is 4 bytes in memory and will be 4 bytes in a disk file written in binary mode. For an ASCII file the size depends on the number of decimal places specified by the [SETPRECISION](#) command

```
DEF myfile as BFILE
DEF iNum as INT
```

```
iNum = 3453423
IF OPENFILE(myfile, "C:\\temp\\temp.bin", "w") = 0
    WRITE myfile, iNum
    CLOSEFILE myfile
ENDIF
```

Would write 4 bytes to the file.

```
DEF myfile as FILE
DEF fNum as FLOAT
fNum = 3453423.3423
IF OPENFILE(myfile, "C:\\temp\\temp.txt", "w") = 0
    WRITE myfile, fNum
    CLOSEFILE myfile
ENDIF
```

The number of bytes written depends on the setting of SETPRECISION. The default is two decimal places. A space character is always written in ASCII mode following a numeric type to separate data and to allow reading a continuous stream of numbers using the READ function.

STRING types are always followed by a carriage return/line feed pair in ASCII mode. To combine separate strings on one line use concatenation or the [APPEND\\$](#) function.

See Also: The [Variables](#) section for information on raw data sizes for all of the data types supported by IW BASIC.

Reading from the file

Writing to files would not be of much use if we could not read what we wrote. The READ function returns 0 on success and has the syntax of:

Error = [READ](#) (file, variable)

file must be a variable of type FILE or type BFILE and have been successfully initialized with the OPENFILE function. *variable* can be any built in type for ASCII files. Binary files also allow direct writing of UDTs.

```
OPENCONSOLE
DEF myfile:FILE
DEF ln:STRING
IF (OPENFILE(myfile, "C:\\IW BASIC TEST.TXT", "R") = 0)
    IF (READ(myfile, ln) = 0)
        PRINT ln
    ENDIF
    CLOSEFILE myfile
    PRINT "File read successfully"
ELSE
    PRINT "File could not be opened"
ENDIF
PRINT "Press Any Key To Close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

When working with ASCII files the trailing carriage return/line feed character is not returned. Numeric data separated by any non numeric character can be read by continually calling the READ function and watching for the end of file with the [EOF](#) function.

```
DEF myfile as FILE
DEF flData as FLOAT
DEF flArray[12,100] as FLOAT
line = 0
IF(OPENFILE(myfile,"C:\\numerics.csv","R") = 0)
  'assume 12 entries per line
  WHILE EOF(myfile) = 0
    FOR x = 1 to 12
      IF READ(myfile, flData) = 0
        'do something with the data
        flarray[x-1, line] = flData
      ENDIF
    NEXT x
    line++
  ENDWHILE
CLOSEFILE myfile
ENDIF
```

When working with binary files it is important to remember that READ will attempt to read as many bytes as required to fill in the variable. For numeric types this is dependant on the size of the type, an INT is 4 bytes for example. For STRING types READ will try and read the full dimension of the string, 255 bytes for normal strings. Arrays will be read/written to the full size of the array regardless of the index specified.

Determining the length of files

The [LEN](#) function works with file variables and returns the length of the open file in bytes.

```
Length = LEN (myfile )
```

Moving the file pointer

The [SEEK](#) command allows setting a binary file pointer to a position for reading or writing, it will not work with ASCII files. SEEK also allows obtaining the current file position.

[SEEK](#) filevariable, position
position = [SEEK](#)(filevariable)

Position specified the zero-based byte offset to begin reading or writing data. The second form of seek returns the current position.

Random Access Files

The READ and WRITE functions are known as sequential file operations. They read and write one piece of data at a time until the end of file is reached. For small files this is fine and very manageable. Lets consider, however, a very large file like an address book. If you have 200

names and addresses stored in a file and want to read the 125th name you would need to read 124 entries before you get to the one you want. Luckily, there is a better way.

The [GET](#) and [PUT](#) statements operate on a binary file using a record number as a parameter. Instead of having to access the 125th entry by reading the entire file you can get to that entry directly. The syntax of GET and PUT is:

[GET](#) filevariable, record, variable

[PUT](#) filevariable, record, variable

File variable must be of type BFILE. The variable to be written can be a user type (UDT) allowing complex records to be used. The record number must be greater than zero.

```
TYPE HighScoreTopTen
    DEF Score as INT
    DEF Time as INT
    DEF pname[20] as ISTRING
ENDTYPE
DEF TopTen as HighScoreTopTen
DEF myfile as BFILE
' ...
IF (OPENFILE(myfile, "C:\\HIGHSCORES.DAT", "W") = 0)
' ...
TopTen.Score = 100
TopTen.Time = 25
TopTen.pname = "The Winner"
PUT myfile, 7, TopTen
```

This would write the 7th record of the file with the contents of TopTen. The number of bytes written for each operation depends on the size of the UDT. You can determine the length of a UDT by using the LEN function with either the name of a defined UDT variable or the typename itself

```
DEF TopTen as HighScoreTopTen
filelen = 10 * LEN(HighScoreTopTen)
```

Copying files

Copies a file from a source directory to a destination. The syntax of COPYFILE is:

error = [COPYFILE](#)(source, dest, fail)

Source and destination are strings containing the full paths of the file to copy. If fail = 1 then the file will not be overwritten if it already exists in the destination directory. If fail = 0 then the file will be overwritten. COPYFILE returns 0 on error.

Deleting files

Deletes the specified file. The syntax of DELETEFILE is:

error = [DELETEFILE](#)(name)

Name is a string containing the full path to the file. [DELETEFILE](#) returns 0 if the file does not exist or is locked by another process.

Creating Directories

[CREATEDIR](#) is used to create a new directory under an existing one.

error = [CREATEDIR](#)(name)

Name is a string containing the full path to the directory. The name should not end with a '\\' character. [CREATEDIR](#) returns 0 on error and cannot create multiple levels of new directories. Each level must be created individually.

```
IF CREATEDIR("c:\\My Programs")
    CREATEDIR("c:\\My Programs\\Samples")
ENDIF
```

Removing directories

[REMOVEDIR](#) is used to remove directories. The directory must be empty before it can be removed. The syntax of [REMOVEDIR](#) is:

error = [REMOVEDIR](#)(name)

Name is a string containing the full path to the directory. The name should not end with a '\\' character. [REMOVE](#) returns 0 on error.

Reading directories

FINDOPEN function

To read all of the file names in a directory use the [FINDOPEN](#) function to get a handle to the directory first. Once a directory is opened the names of the files can be obtained with the [FINDNEXT](#) function. The syntax of [FINDOPEN](#) is:

handle = [FINDOPEN](#)(directory)

Directory is a string that contains the full path to the directory plus any wildcard symbols for file matching. Example: "c:\\windows*.txt". *handle* is an integer variable. [FINDOPEN](#) returns 0 if the directory could not be opened for reading.

FINDNEXT function

After a directory is successfully opened with the [FINDOPEN](#) function use [FINDNEXT](#) to retrieve the filenames in a loop. [FINDNEXT](#) returns an empty string when all of the file names have been retrieved, an optional attribute parameter returns the files attributes. The syntax of [FINDNEXT](#) is:

name = [FINDNEXT](#)(handle {, attrib})

Handle is the integer value returned by FINDOPEN.

For a list of possible returned attributes see [FINDNEXT](#) in the alphabetical reference.

FINDCLOSE statement

After you are finished reading a directory you must close the handle with [FINDCLOSE](#). If the handle is not closed memory loss will occur. The syntax of FINDCLOSE is:

[FINDCLOSE](#) handle

You must not close a handle more than once.

Example:

```
OPENCONSOLE
DEF dir:INT
DEF filename:STRING

dir = FINDOPEN("c:\\*. *")
IF(dir)
  DO
    filename = FINDNEXT(dir)
    PRINT filename
    UNTIL filename = ""
    FINDCLOSE dir
  ENDIF

PRINT "Press Any Key"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Opening the system file dialog

[FILEREQUEST](#) opens a standard file dialog and returns a string containing the fully qualified pathname to the file.

Name\$ = [FILEREQUEST](#) (prompt, parent, type {, filter} {,ext} {,flags} {,Initial Directory})

If *type* equals 1 then an 'Open' dialog is used. If *type* equals 0 then a 'Save As' dialog is opened. The optional *filter* variable limits the dialog to showing only certain file types. The string consists of ordered pairs separated by the '|' character and ending with two ||.

Example:

```
Filter$ = "Text files|.txt|All Files|. *||"
```

A default extension can be supplied in the optional *ext* parameter and will be used if the user does not type in an extension. This is a string parameter and should not contain the '!'

To allow users to select multiple files supply @MULTISELECT for the *flags* parameter. The returned string will contain complete paths to all of the files selected separated with the '|' character. If only one file is selected it will be terminated with the '|' character. The following

example shows how to extract the filenames from the returned string:

```
REM Define a buffer to hold the returned filenames.
DEF filenames[10000]:ISTRING
DEF filter,filetemp:STRING
DEF pos:int
filter = "All Files (*.*)|*.*|Text Files (*.txt)|*.txt||"

filenames = filerequest("Select Multiple Files",0,1,filter,"txt",@MULTISELECT, "C:\\")
do
  pos = instr(filenames,"|")
  if(pos)
    filetemp = left$(filenames,pos-1)
    filenames = mid$(filenames,pos+1)
    REM do something with the file in filetemp
  endif
until pos = 0
```

9.3 Formatting Output

While its easy enough to use PRINT and SETPRECISION to control how numbers are displayed in a window, or console. Sometimes more advanced formatting is necessary. The USING/WUSING functions return a string that is formatted based on a specifier string and one or more parameters. The syntax of the USING function is:

```
USING( formatstring, param1 {,param2...} )
WUSING( formatstring, param1 {,param2...} )
```

The *format string* is a string literal or variable that can contain special formatting symbols as well as regular text to be inserted into the final output string. USING understands the following format specifiers:

Symbol	Meaning
#	Reserves a place for one digit
Point (.)	Determines decimal point locations
Minus (-)	Left justifies within field. Default is right.
0	Prints leading zeros instead of spaces. Ignored if used with left justification.
Comma (,)	Prints a comma before every third digit to the left of the decimal point and reserves a place for one digit or digit separator.
&	Copies the string parameter directly
%d	Treat the parameter as a DOUBLE
%f	Treat the parameter as a FLOAT
%q	Treat the parameter as a 64 bit integer (INT64)
%%	Inserts a % sign into the output string

An example format string would look like "\$#,###.##" which would reserve two places to the right of the decimal place and four to the left. The result string would also have comma's inserted between every third and fourth digit. The output can be used as a parameter to any function that accepts a string, assigned to a string variable or used with the PRINT statement.

```
PRINT USING("$#,###.##", 1145.551)
A$ = USING("$#####.##", 22.8)
PRINT A$
```

Would produce the output of:

```
$1,145.56
$ 22.80
```

Note that the '\$' is copied to the result string directly. The comma does not need to be placed correctly in the format string. It just needs to be somewhere in the definition.

```
PRINT USING("#,#####", 1234567)
```

Produces the output of:

```
1,234,567
```

Truncation and rounding

USING will not truncate the output if the number of digits exceeds the number of # symbols on the left of the decimal point. Make sure you have enough # symbols to accommodate the output width. The right side of the decimal point is always rounded and truncated to match the format definition.

```
PRINT USING("##.###", 5115.1234)
```

Produces the output of:

```
5115.123
```

Filling with spaces or 0

The left side of the decimal point determines how many spaces or 0's to use as a fill value if there are not enough digits to fill the field.

```
PRINT USING("0#####", 23)
PRINT USING("#####", 23)
```

Produces the output of:

```
000023
 23
```

Justification

To left justify the output in the field use a minus sign (-) as a leading character.

```
PRINT USING("-#### -####", 55, 88)
```

Produces the output of:

```
55    88
```

Including string variables

To copy the contents of a string literal or variable use the & symbol in your definition

```
PRINT USING("&$#,###.##&", "The total cost is ", 3000.55, " Dollars")
```

Produces the output of:

```
The total cost is $3,000.55 Dollars
```

Expected types and overriding defaults

USING expects certain variable types to appear in the parameter list depending on the contents of the formatting string. If the formatting string contains a decimal point then the default is a DOUBLE type. If no decimal point is specified then USING expects an INT variable type. To override the expected defaults use the %.. format specifier to inform USING what type of data you are including. USING will convert the data appropriately as needed to match the formatting string

```
'%d is needed because there is no decimal point in the format string
'and we are sending a DOUBLE type
PRINT USING("%d#####", 123.5567)
DEF flNum as FLOAT
flNum = 1.23456f
' %f is needed since we are sending a FLOAT type
PRINT USING("%f###.###", flNum)
```

9.4 Using Strings

Strings are probably the most used variable types in programs. A string in IWBASIC is an array of characters terminated by a single NULL character (0). Strings can be defined as variables with the DEF / DIM statement, defined as literals by enclosing text in quotes, or returned from functions.

The STRING variable type is dimensioned automatically to 255 bytes which is enough room for 254 characters and the terminating NULL. The ISTRING variable type is unique to the IWBASIC language and can be dimensioned to create a string from 1 to the amount of available memory.

```
DEF mystring as STRING
DEF bigstring[25000] as ISTRING
```

Strings can be loaded, or initialized by assigning with the = operator.

```
mystring = "This is a string"  
bigstring = STRING$( "A", 24999)
```

The WSTRING variable type is dimensioned automatically to 255 Unicode characters which is enough room for 254 characters and the terminating double NULL. The IWSTRING variable type can create a Unicode string of any size, up to the amount of available memory.

```
DEF mystring as WSTRING  
DEF bigstring[25000] as IWSTRING
```

Unicode strings can also be loaded, or initialized by assigning with the = operator.

```
mystring = L"This is a string"  
bigstring = WSTRING$(L"A", 24999)
```

NOTE: The compiler can not check for overwriting the ends of dimensioned strings so make sure you check your lengths with the [LEN](#) function, or otherwise limit the amount of text assigned to a string.

Common string functions

Included with the standard command set are a number of string functions for creating, manipulating and working with strings.

APPEND\$ function

Concatenates all the strings in the argument list and returns the total string. This is similar to using the '+' operator on strings.

Result\$ = **APPEND\$** (string1, string2 {,stringn ...})

See also: [Operators](#) , concatenation with the + operator

ASC function

The ASC function returns the ASCII value of the character argument. A string may be specified as the argument which case the first character of the string is used.

Value = **ASC**("A")

CHR\$/WCHR\$ function

Returns the character represented by the ASCII parameter. This is the opposite of the ASC function. This function is useful for creating characters that can not be normally typed on a keyboard. The syntax of the CHR\$ function is:

Character = **CHR\$**(value)

Example print a <RETURN> to the console:

```
PRINT CHR$(13)
```

The WCHR\$ function accepts a word value and returns a character as a Unicode string.

DATE\$ function

Returns the current date as a string in the format DD-MM-YYYY. The syntax of the DATE\$ function is:

```
Result$ = DATE$
```

DATE\$(format) function

Returns the current date as a string formatted by a specifier string. The specifier string is comprised of:

Specifier	Result
d	Day of month as digits with no leading zero for single-digit days.
dd	Day of month as digits with leading zero for single-digit days.
ddd	Day of week as a three-letter abbreviation.
dddd	Day of week as its full name.
M	Month as digits with no leading zero for single-digit months.
MM	Month as digits with leading zero for single-digit months.
MMM	Month as a three-letter abbreviation.
MMMM	Month as its full name.
y	Year as last two digits, but with no leading zero for years less than 10.
yy	Year as last two digits, but with leading zero for years less than 10.
yyyy	Year represented by full four digits.

For example, to get the date string

"Wed, Aug 31 94"

use the following input string:

"ddd', ' MMM dd yy"

Example:

```
PRINT DATE$("ddd', ' MMM dd yy")
```

Note the single quotes used to insert text into the output string.

HEX\$/WHEX\$ function

Converts the numeric parameter to a string representing the hexadecimal notation of that number.

The syntax of HEX\$ is:

Result\$ = HEX\$(value)

```
PRINT HEX$(255)
```

Would print FF to the console window. HEX\$ uses a UINT64 parameter and can display hex numbers up to 64 bits in length. The WHEX\$ function is the Unicode version.

INSTR/WINSTR function

The INSTR function is used to search for text in a string.

Position = `INSTR(string1, string2 {,start})`

Returns the position of the sub string *string2* in *string1*. Or returns 0 if *string2* is not in *string1*. Optional *start* variable specifies a starting point in *string1* to begin searching. *position* and *start* are ones based. The WINSTR function works with Unicode strings.

INSTR performs a case sensitive search. To perform a case insensitive search use either LCASE\$ or UCASE\$ on the with the parameters.

```
IF INSTR(UCASE$(mystr), "FOX") > 0
    PRINT "Fox was found in mystr"
ENDIF
```

LCASE\$/WLCASE\$ function

Returns the string parameter converted to all lowercase letters.

```
lwr$ = LCASE$("CONVERT THIS TO LOWER CASE")
PRINT lwr$
```

LEFT\$/WLEFT\$ function

Extracts count number of characters from the string starting from the leftmost character. Returns the resulting string. The syntax of LEFT\$ is:

```
str = "The red fox"
Result$ = LEFT$(str, 3)
PRINT Result$
```

LEN function

The LEN function works with any variable type in IW BASIC. For strings it returns the number of characters in the string not including the NULL terminator.

```
MyString = "What's my length?"
PRINT LEN(mystring)
```

LTRIM\$/WLTRIM\$ function

Removes all leading white-space characters from a string. White-space characters include spaces

and tabs. LTRIM\$ returns the result as a string leaving the parameter unchanged. The syntax of LTRIM\$ is:

Result\$ = LTRIM\$(string)

MID\$/WMID\$ function

Extracts count number of characters starting at position from a string. If *count* is omitted all of the characters from position to the end of the string are returned.

Result\$ = MID\$(str, position {,count})

If *count* is omitted all of the characters from position to the end of the string are returned. *position* is ones based.

```
MyString = "Extract some characters from me"  
PRINT MID$(MyString, 8, 4)
```

RIGHT\$/WRIGHT\$ function

Extracts characters from the end of a string. The syntax of the RIGHT\$ function is:

Result\$ = RIGHT\$(string, count)

Count is the number of characters to extract from the end of the string.

```
MyString = "Extract some characters from me"  
PRINT RIGHT$(MyString,2)
```

RTRIM\$/WRTRIM\$ function

Removes any trailing white-space characters from a string. White-space characters include spaces and tabs. RTRIM\$ returns the result as a string leaving the parameter unchanged. The syntax of RTRIM\$ is:

Result\$ = RTRIM\$(string)

SPACES\$/WSPACES\$ function

Returns a string filled with n spaces. The syntax of the SPACES\$ function is:

Result\$ = SPACES\$(n)

STR\$/WSTR\$ function

The STR\$ function returns the string representation of a number. The syntax of STR\$ is:

Result\$ = STR\$(number)

number is a double precision number and the conversion will contain as many decimal places as specified by the [SETPRECISION](#) command.

```
num$ = STR$(1.23456)
PRINT num$
```

STRING\$/WSTRINGS\$ function

The STRING\$ function returns a string filled with count number of the character specified. The syntax of the STRING\$ function is:

Result\$ = **STRING\$** (count, character)

TIMES\$ function

Returns the current system time in the format HH:MM:SS as a string. The syntax of TIMES\$ is:

Result\$ = **TIMES\$**

UCASE\$/WUCASE\$ function

Returns the string parameter converted to all uppercase letters. The syntax of the UCASE\$ function is:

Result\$ = **UCASE\$(** string)

VAL/WVAL function

Returns the value of a string representation of a number. This function is the opposite of the STR\$ function. The syntax of the VAL function is:

Result = **VAL**(string)

The return type is a double precision number.

```
DEF d as DOUBLE
d = VAL("1.234567")
```

VAL will stop converting digits on the first non numeric character. Use LTRIM\$ on your variable to remove any leading whitespace before conversion.

REPLACE\$ statement

The REPLACE\$ statement replaces characters in one string with one or more characters from another. The syntax of the REPLACE\$ statement is:

REPLACE\$ dest, start, count, source

Dest is the string being modifies and source is the string where the characters are extracted from. Start and count must be greater than 0.

Example:

```
DEF s:string
s = "All good DOGS go to heaven"
REPLACE$ s,10,3,"dog"
PRINT s
```

Would print "All good dogs go to heaven"

Converting between Unicode and ANSI

Converting between a wide character (Unicode) and ANSI strings is accomplished by using the W2S and S2W functions. W2S accepts a Unicode string as input and returns an ANSI string as output. S2W accepts an ANSI string as input and returns a Unicode string as output.

```
DEF s:STRING
DEF w:WSTRING
s = W2S(L"This is a Unicode string")
PRINT s
w = WLEFT$(L"All good pets deserve love",8)
PRINT W2S(w)
```

9.5 Writing DLL's

IWBASIC can create DLL's for use in any language. A DLL is really just a collection of subroutines that are exported so other processes can use them.

The EXPORT statement

Creating the DLL is as simple as writing one or more subroutines adding an [EXPORT](#) statement for each subroutine that you want visible and compiling as a DLL target. In both project and single file mode the default extension is .dll and really should not be changed.

A simple DLL example:

```
export myfunction
export myfunction2
export INTRAND

SUB myfunction(in as INT),INT
RETURN in * 100
ENDSUB

SUB myfunction2(),INT
RETURN 0
ENDSUB

SUB INTRAND(min as INT,max as INT),INT
RETURN RAND(min,max)
ENDSUB
```

If you wish to use your newly created DLL with IWBASIC just create an import library for it as outlined in [Using DLL's](#). The DLL must be either in your system directory or the executables directory.

When creating a DLL you can change the preferred load address, also known as the base address, by clicking on the "Advanced" button of the executable options or project options dialog.

Windows uses the base address to determine where to load the DLL and if you are using more than one DLL with the same base address then conflicts can arise.

The default base address for DLL's is 0x10000000 hex.

Notes

If you wish your functions to use the CDECL calling convention, like that in the C runtime library, you must declare your function as in:

```
export INTRAND
DECLARE CDECL INTRAND(min as INT,max as INT),INT
SUB INTRAND(min as INT,max as INT),INT
RETURN RAND(min,max)
ENDSUB
```

9.6 Using resources

Resources are files and data that get compiled with your application and embedded into the executable. Resources can only be used with projects and are not available for single file compiling.

Resources are displayed, edited, added, and deleted in the [Resources](#) tab of the [Output Window](#). See that section for additional information as well as the [How-To»Projects»Resources](#) section.

Resource functions

success = [LOADRESOURCE](#)(ID, Type, Variable)

Loads a resource from the executable and places either a copy of the resource, or pointer to the resource, in the variable specified. [LOADRESOURCE](#) is most useful for reading raw data with [@RESDATA](#). For normal loading of images, icons and cursors you should use the [LOADIMAGE](#) function.

ID is either a numeric or string identifier to the resource, *TYPE* is a numeric or string type and it stores the info in *variable*. The standard Windows resource types can be specified and loaded in raw form using the following constants:

```
@RESCURSOR
@RESBITMAP
@RESICON
@RESMENU
@RESDIALOG
@RESSTRING
@RESACCEL
@RESDATA
@RESMESSAGETABLE
@RESGROUPCURSOR
@RESGROUPICON
@RESVERSION
```

If *variable* is a STRING (or ISTRING) variable the contents of the resource will be copied into the string. Useful for embedded text resources.

If *variable* is an INT or UINT variable then a handle to the locked resource is returned. Useful for API calls.

If *variable* is a POINTER type then a pointer to the locked resource is returned. The data can be accessed directly with type casting and dereferencing.

If *variable* is a MEMORY variable then memory is allocated for the resource and the resource is copied into it. You can then use READMEM to read the resource data. LEN(*variable*) will return the length of the resource in this case. You must free the memory returned in this case with the FREEMEM statement. Only use a newly defined MEMORY variable or one that has been freed with FREEMEM. Using a MEMORY variable that has been allocated with ALLOCMEM will result in memory leaks in your program.

handle = LOADIMAGE (filename | resourceID, type)

Loads an image from the resources. Type is a numeric value defining what kind of image to load.

The valid values for type are:

@IMGBITMAP - bitmap (*.bmp)

@IMGICON - Icon (*.ico)

@IMGCURSOR - Cursor (*.cur)

@IMGSCALABLE - scalable bitmap, JPEG (*.jpg) or GIF (*.gif) files.

@IMGOEM - OR '|' in with @IMGBITMAP, @IMGICON or @IMGCURSOR to load an OEM (system) resource.

If a filename is specified the image is loaded from disk. LOADIMAGE can also load bitmaps, icons and cursors directly from the executables resources. Enhanced meta files cannot be loaded from the resource table. Scalable images must be added as a 'Scalable Image' in the resource add dialog. Internally this is saved as type "RTIMAGE" if you need to use LOADRESOURCE to access the image data directly.

Resource ID is either the string or integer identifier of an image resource compiled with the project.

When you are finished using an image remember to free the image with the [DELETEIMAGE](#) command.

success = LOADMENU(window | dialog, resourceID)

Loads a menu definition from resources and creates the menu bar for the window or dialog specified. *ResourceID* is the string or integer identifier of the menu. The return value is 0 if the menu could not be loaded from the resources.

For example a menu definition may look like this:

```
BEGIN
  POPUP "&File"
```

```

BEGIN
    MENUITEM "&New\tCtrl+N", 1
    MENUITEM "&Open...\tCtrl+O", 2
    MENUITEM SEPARATOR
    MENUITEM "P&rint Setup...", 3
    MENUITEM SEPARATOR
    MENUITEM "Recent File", 4
    MENUITEM SEPARATOR
    MENUITEM "E&xit", 5
END
POPOP "&Help"
BEGIN
    MENUITEM "&About ChartCraft...", 6
END
END

```

length = GETRESOURCELENGTH(resourceID, type)

Returns the length of the specified resource in bytes. When extracting a resource for saving to disk it is best to use this function to return the exact length.

9.7 MIDI Music and Sound

IW BASIC supports playing wave files and MIDI streams natively.

Playing wave files

To play a wave sound from either memory or a file use the PLAYWAVE function. The sound can be played either synchronously or asynchronously. Synchronous playback will stop your program until the sound is finished playing. The syntax of PLAYWAVE is:

```
success = PLAYWAVE(filename | memory, flags)
```

Valid values for *flags* are:

@SNDASYNC - The sound is played asynchronously and the function returns immediately after beginning the sound.

@SNDSYNC - The sound is played synchronously and the function does not return until the sound ends.

@SNDLOOP - The sound plays repeatedly. You must also specify @SNDASYNC.

@SNDNOSTOP - If a sound is currently playing, the function immediately returns FALSE, without playing the requested sound.

Example:

```
PLAYWAVE "c:\\media\\bark.wav", @SNDASYNC
```

To stop a sound use an empty string for the name or use NULL. Playing from resources can be accomplished by using a custom resource and LOADRESOURCE to load the wave resource directly into a MEMORY variable.

Playing MIDI streams

A MIDI stream is a sequence of notes played either synchronously or asynchronously. The PLAYMIDI\$ function provides an easy way to create music for your program. PLAYMIDI\$ uses a formatting string describing the notes to be played and the properties of the notes including length, instrument, and velocity. The string is interpreted and converted into the necessary MIDI events for the stream before playing. The syntax of PLAYMIDI\$ is:

```
pThread = PLAYMIDI$( strMidi, OPT bAsync)
```

strMidi is the note formatting string and can contain the following commands:

A-G {#}n	Plays a note. The # specifies a sharp. <i>n</i> is the duration of the note and can be 1,2,4,8,16,32 or 64 representing whole note, half note, quarter note, eighth note, and so on.
On	Specifies the octave of the notes that follow. There are 11 octaves from 0 to 10 with octave 5 containing middle C. Each octave begins with the C note and ends with the B note.
In	Sets the instrument (patch) to be used for the notes that follow. <i>n</i> can be from 0 to 127.
Vn	Sets the velocity, or volume, of the notes that follow. <i>n</i> ranges from 0 which is completely silent to 127 for full velocity.
Nn	Sets the channel of the notes that follow. <i>n</i> is the channel number and can range from 0 to 15. The default channel is 0. Channel 9 is the built in drum channel for most sound cards.
Rn	Inserts a rest. <i>n</i> specifies the duration of the rest and are the same as note durations.
Tn	Sets the tempo of the music in beats per minute (bpm). Default is 120. <i>n</i> can range from 10 to 300
*	Begins a chord. Any commands between the * and ; are played at the same time index.
;	Ends a chord.

Any other characters in the formatting string are simply ignored allowing separating commands with spaces, tabs or whatever is convenient. The commands are not case sensitive.

Parameters are saved between note commands so it is not necessary to specify the duration, octave, velocity, etc for every note. You only need to specify the numeric parameter when there is a change.

Examples:

```
PLAYMIDI$ "T120 N0 I25 O5 *C2EG;D4EFG*A2O6CE; *C2EG;D4EFG*A2O7CE;"
PLAYMIDI$ "T180 N0 I0 O5 C8C#DD#EFF#GG#AA#B O6 CC#DD#EFF#GG#AA#B O7 C1"
```

To play music asynchronously specify TRUE for the optional *bAsync* parameter. The command will return immediately and the notes will be played in the background while your program continues. PLAYMIDI\$ returns a pointer to the stream thread being played that can be passed to the STOPMIDI\$ command. Only one asynchronous stream is allowed at a time by Windows so it will be necessary to stop the previous stream with STOPMIDI\$ before starting another one.

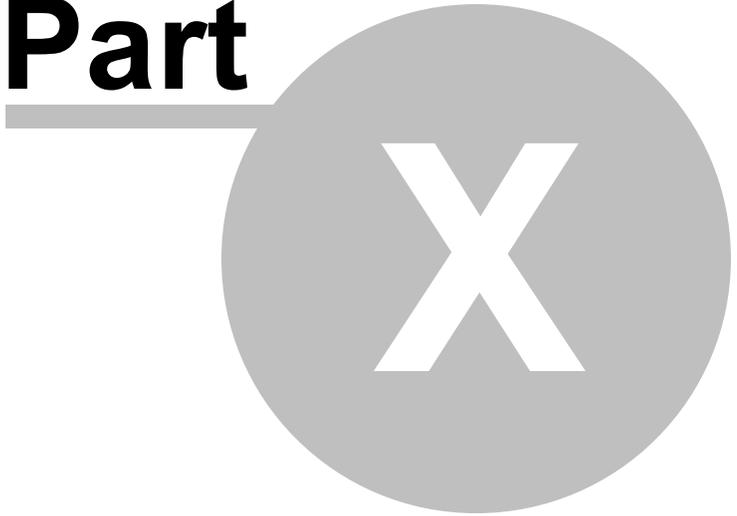
Example:

```
OPENCONSOLE
pThread = PLAYMIDI$( "T180 N0 I0 O5 C8C#DD#EFF#GG#AA#B O6 CC#DD#EFF#GG#AA#B O7 C1", TRUE, TRUE)
DO:UNTIL INKEY$ <> ""
STOPMIDI$(pThread)
PLAYMIDI$ "T120 N0 I25 O5 *C2EG;D4EFG*A2O6CE; *C2EG;D4EFG*A2O7CE;"
```

It is not necessary to stop an asynchronous music stream before your program ends. The compiler adds the necessary commands to stop any running streams on exit.

Windows Programming

Part



10 Windows Programming

10.1 Creating a Window

Opening a Window

Opening a window in IWBASIC is easily done with the [OPENWINDOW](#) function. When you use IWBASIC to write Windows based software your program becomes *event driven*. Every action a user takes when running your program is sent to a special subroutine as a message. Your program must then decide whether or not to respond to this message. A message is really just a number. This number represents an action taken by the user.

The syntax of the OPENWINDOW function is:

OPENWINDOW variable, left, top, width, height, flags, parent, title, handler

Parameters:

variable - The name of the WINDOW variable used to store the window

left - The left edge of the window

top - The top edge of the window

width - The width of the window

height - The height of the window

flags - A numeric value specifying creation style flags

parent - a WINDOW variable if this is a child window or NULL

title - The text shown in the caption of the window

handler - The address of a subroutine to handle messages for the window. Use the & operator.

The OPENWINDOW command will return 1 if the window was created successfully or 0 if the window could not be created. You can also check the window variable to see if it equals 0. If so then the window could not be opened.

Example:

```
REM define a window variable
DEF w1 as WINDOW
REM open the window
OPENWINDOW w1,0,0,350,350,@MINBOX|@MAXBOX|@SIZE,NULL,"Simple Window",&main
REM print a message
PRINT w1,"Hello World"
REM when w1 = 0 the window has been closed
WAITUNTIL w1 = 0
END

REM every time there is a message for our window
REM the operating system will GOSUB here
SUB main( ), INT
  IF @MESSAGE = @IDCLOSEWINDOW
    REM closes the window and sets w1 = 0
```

```

        CLOSEWINDOW w1
    ENDF
RETURN 0
ENDSUB

```

The example shows the steps necessary to open a window and wait for a message. Windows will call your handler subroutine anytime there is a message for the window you created.

System variables and constants

The previous example also introduces the '@' symbol. IW BASIC defines a number of constants for use in your program. Special variables contain information about the message sent to a window. In either case the '@' symbol is used to differentiate these names from variables you define in your program. System variables like @MESSAGE are set by IW BASIC and cannot be used as normal variables. You can define your own constants with the [SETID](#) statement.

The appendix contains a list of all of the system variables and their meanings.

Creation style flags

In the above example we used @SIZE in the flags parameter. This tells IW BASIC that the window should be resizable. More than one flag can be used and combined with the '|' symbol (meaning OR). IW BASIC contains many predefined flags:

NAME	PURPOSE/STYLE
@SIZE	Creates a window that is resizable
@MINBOX	The window has a minimize box
@MAXBOX	The window has a maximize box
@MINIMIZED	Creates a window that is initially minimized
@MAXIMIZED	Creates a window that is initially maximized
@CAPTION	Default. Creates a window with a caption
@NOCAPTION	Creates a window with no caption
@SYSMENU	Default. Creates a standard system menu
@BORDER	Creates a bordered window. Use with the @NOCAPTION flag
@HSCROLL	Window has a horizontal scroll bar
@VSCROLL	Window has a vertical scroll bar
@MDIFRAME	Creates a frame window that can contain child client windows.
@USEDEFAULT	Child windows can use this for the 'left' parameter to let windows pick a default size for the window
@TOOLWINDOW	Creates a window with a half sized caption.
@NOAUTODRAW	Allows your program to handle @IDPAINT messages
@TOPMOST	Creates a window that stays on top of all others. Cannot be used with MDI child windows.

@AUTOSCALE	Creates a window that scales the contents automatically
@FIXEDSIZE	Use to create an autodrawn window that will remain the same size throughout the life of the window.
@HIDDEN	Creates a window that is initially hidden.

Waiting for messages

In the previous example, we introduced a few new ideas. The [WAITUNTIL](#) command tells IWBASIC to process messages until a condition is true. The WAITUNTIL command *sleeps* until something is done with one of the windows your program is using. When a message is sent to your window, the handler subroutine is called that was defined when the window was opened. After your subroutine executes a RETURN statement, your program will continue waiting for messages. In our example, we check the value of the variable 'w1' to see if we should wait for more messages or end the program. The syntax of WAITUNTIL is:

[WAITUNTIL](#) condition

IWBASIC also contains another statement for processing messages. The WAIT statement can be used when more control over message processing is desired. WAIT processes any messages that are available for your window, sleeps if none are available, and then returns. The syntax of WAIT is:

[WAIT](#) {NoSleep}

If the optional *NoSleep* parameter is set to 1 then the WAIT command will check for messages and return immediately. WAIT will be covered in more detail in the [Messages and Message loops](#) section

Handling Messages

When your windows handler subroutine is called, the system variable @MESSAGE will contain the message ID. This variable can then be compared against any of the messages you wish to handle using an IF or SELECT statement. Some messages contain additional information and set @WPARAM and @LPARAM accordingly. A good example of this is information from the keyboard. When a key is pressed your window will receive, among others, the message @IDCHAR. In this case @WPARAM will contain the ASCII value of the key.

For backwards compatibility with other languages you can use @CLASS, @CODE and @QUAL in place of @MESSAGE, @WPARAM and @LPARAM respectively.

Example code fragment:

```
SUB mywin( ), INT
SELECT @MESSAGE
    CASE @IDCHAR
        Key = @WPARAM
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
```

```
RETURN 0
ENDSUB
```

When developing your handler subroutine you can handle as many or as few messages as you need. Any messages that you do not respond to are simply 'thrown away' by the system. At a minimum you should check for @IDCLOSEWINDOW to handle the close button of the window.

Returning Values

If a particular message requires a return value you can specify it in the RETURN statement.

The next section will explore messages and the message loops in more detail.

See Also: [System Variables and Constants](#) in the appendix for a list of message IDs

10.2 Messages and Message Loops

Introduction

In the previous section we discussed how to create a window and introduced the concept of *messages*. Any action taken by a user that effects a window or dialog is reported to your program. This report is sent as a message that contains three main pieces of information. The message class or ID, the message code also known as the WPARAM, and the message qualifier also known as the LPARAM.

Windows sends the report to your program by calling the subroutine you specified when the window or dialog was created. This subroutine is known as the "handler" subroutine. In other languages it may be referred to as the "windows procedure". Before your handler subroutine is called, Windows sets the system variables @MESSAGE, @WPARAM and @LPARAM so your program can tell what has happened.

The message queue and loop

Since Windows is a multitasking system, and there may be literally hundreds of programs running on your computer at once, it would be very inefficient for Windows to wait for your program to finish handling a message. So all messages sent to your program are placed in a special buffer known as the message queue. The messages are placed in the queue in the order received to prevent your program from losing track of what has been happening to the window. The queue also prevents losing messages that may have been sent while your program was busy doing something else.

Because there may be many other programs running at the same time it is also not system friendly to "busy wait" for messages. When there are no messages left to process on the queue your program should sleep, waiting for the next message to be reported to the window, and placed in the message queue. In IWBASIC the message queue, looping and sleeping are handled by using either the [WAIT](#) or [WAITUNTIL](#) statements.

The WAITUNTIL statement.

As discussed in the section on creating windows the WAITUNTIL statement will be used almost exclusively in your programs. To fully understand what is happening when you use the WAITUNTIL statement we will use some IW BASIC statements to break it down. **This code is just an example of what WAITUNTIL does and is not an actual program:**

```

SUB WAITUNTIL(condition)
DO
SLEEP(): REM Wait until some messages are available
FOR x = 0 TO GetNumMessages() : REM get the number of messages in the queue
  msg = GETMESSAGE(x)
  @MESSAGE = msg.messageID
  @WPARAM = msg.wparam
  @LPARAM = msg.lparam
  IF @MESSAGE = @IDCONTROL
    @CONTROLID = msg.control_id
  ENDIF
  IF @MESSAGE = @MENU PICK
    @MENUNUM = msg.menu_id
  ENDIF
  GOSUB msg.window.handler : REM call the handler subroutine for this window or dialog
NEXT x
UNTIL condition
RETURN
ENDSUB

```

As you can see by the pseudo code above there actually is a loop comprised of the DO and FOR statements.

The WAIT statement

The WAIT statement is very similar to WAITUNTIL with the exception that it only executes once. It is up to your program to determine how long messages should be processed and what condition (s) to check to exit the loop. WAIT should only be used in special circumstances where you need more control of when messages are processed. The WAIT statement has an optional parameter to specify not to sleep and to immediately check for messages and return. Using pseudo IW BASIC statements again the code for WAIT would look something like this:

```

SUB WAIT(nosleep)
IF nosleep <> 1
  SLEEP(): REM Wait until some messages are available
ENDIF
FOR x = 0 TO GetNumMessages() : REM get the number of messages in the queue
  msg = GETMESSAGE(x)
  @MESSAGE = msg.messageID
  @WPARAM = msg.wparam
  @LPARAM = msg.lparam

```

```
IF @MESSAGE = @IDCONTROL
    @CONTROLID = msg.control_id
ENDIF
IF @MESSAGE = @MENUPICK
    @MENUNUM = msg.menu_id
ENDIF
GOSUB msg.window.handler : REM call the handler subroutine for this window or dialog
NEXT x
RETURN
```

The purpose of having a nosleep parameter is to allow your program to remain responsive to user input while it is very busy doing something else. For example:

```
FOR x = 1 TO 10000000
    '...some complex math and drawing here
    WAIT 1
    IF cancel = 1 THEN x = 10000001
NEXT x
```

This assumes that there is some menu option, or button that sets the variable cancel equal to 1. If you did not use WAIT 1 in this case the program would appear to be 'locked' and would not respond to menus or buttons until the long loop was finished. You should limit your use of WAIT 1 since this creates a "busy wait" loop. Since your program is never allowed to sleep it consumes the majority of the processor time on the system and this will reduce overall system performance.

WAIT can also be used to create a custom WAITUNTIL loop:

```
SUB MYWAITUNTIL
DO
    processdata()
    WAIT
UNTIL (cancel = 0) | (run = 0)
RETURN
ENDSUB
```

Using WAIT in this manner allows using the time between messages for custom processing.

Message ID's and the handler

Now that you have a general idea of what is happening with the message queue and loop we can explore some of the message ID's your handler will receive. To review the handler is just a subroutine that is called by the system, through either the WAITUNTIL or WAIT statements. IW BASIC predefines many of the common message ID's that your program will use. Windows defines many hundreds more that can be used with your IW BASIC programs. A good source of information on Windows messages is the Windows SDK available from Microsoft, the windows header files, or from the [Microsoft developers website](#)

Mouse messages

The mouse generates an input events whenever the user moves the mouse, or presses or releases a mouse button. Windows converts mouse input events into messages and posts them to the appropriate programs message queue. When mouse messages are posted faster than a program can process them, Windows discards all but the most recent mouse message.

A window receives a mouse message when a mouse event occurs while the cursor is within the borders of the window, or when the window has captured the mouse. Mouse messages are divided into two groups: client area messages and nonclient area messages. Typically, an application processes client area messages and ignores nonclient area messages.

When a mouse message is received and your handler is called the system variables @MOUSEX and @MOUSEY will contain the position of the pointers hot spot at the time the message was generated. The following mouse message ID's are predefined in IWBASIC

Message ID	Meaning	Windows equivalent
@IDMOUSEMOVE	The mouse was moved in the client area	WM_MOUSEMOVE
@IDLBUTTONDOWN	Left mouse button was pressed while the pointer was in the client area	WM_LBUTTONDOWN
@IDLBUTTONUP	Left mouse button was released while the pointer was in the client area	WM_LBUTTONUP
@IDLBUTTONDBLCLK	Left mouse button was double clicked while the pointer was in the client area	WM_LBUTTONDBLCLK
@IDRBUTTONDOWN	Right mouse button was pressed while the pointer was in the client area	WM_RBUTTONDOWN
@IDRBUTTONUP	Right mouse button was released while the pointer was in the client area	WM_RBUTTONUP
@IDRBUTTONDBLCLK	Right mouse button was double clicked while the pointer was in the client area	WM_RBUTTONDBLCLK

Additional information:

@WPARAM contains the status of the other mouse buttons and the CTRL/SHIFT keys when the message was received.

0x0001 = left mouse button is down

0x0002 = right mouse button is down

0x0004 = SHIFT is being held down

0x0008 = CTRL is being held down
 0x0010 = middle mouse button is down

The values should be test by using a bit wise AND (&). For example:
 IF (@WPARAM & 0x0004) = 0x0004 : REM shift key is held down

Keyboard messages

Messages from the keyboard are generated whenever a key is pressed while your window has focus. Keyboard messages are sent in two different forms, keystroke messages and character messages. Keystroke messages are sent as raw keyboard scan codes, before the system translates them. Character messages are sent when the system translates the raw key code, taking into account the SHIFT, ALT and CTRL keys. The following keyboard messages ID's are predefined by IW BASIC:

Message ID	Meaning	Windows equivalent
@IDKEYDOWN	A key was pressed while the window had focus	WM_KEYDOWN
@IDKEYUP	A key was released while the window had focus	WM_KEYUP
@IDCHAR	A keystroke or combination generated a valid ASCII character.	WM_CHAR

Additional information:

@WPARAM will contain the raw virtual keycode for @IDKEYDOWN and @IDKEYUP messages or the ASCII character for @IDCHAR messages. See [Appendix C](#) for a list of virtual key codes.

Window and system messages

Messages for your window are generated whenever an action that would effect the window happens. These include sizing messages, creation messages, drawing messages, close events, system wide messages, etc. A few of the predefined system messages:

Message ID	Meaning	Windows equivalent
@IDCREATE	Sent when the window is first created but before it is displayed	WM_CREATE
@IDDESTROY	Sent when the window is about to be destroyed.	WM_DESTROY
@IDINITDIALOG	Sent when the dialog is about to be shown. Initialize all controls here	WM_INITDIALOG
@IDCLOSEWINDOW	The close button was pressed in a window or modeless dialog	WM_CLOSE

@IDMOVE	Sent when the window had been moved by dragging the title bar.	WM_MOVE
@IDMOVING	Send while the window is moving by dragging the title bar.	WM_MOVING
@IDSIZE	Sent when the window or dialog is being sized or has been resized. This is the combination of two messages.	WM_SIZE WM_SIZING
@IDSIZECHANGED	Sent when the window or dialog has been resized	WM_SIZE
@IDSIZING	Sent when the window or dialog is being sized	WM_SIZING
@IDERASEBACKGROUND	Sent when the background of a window needs to be painted. The window background is shown when transparent objects, such as toolbars with the @TBFLAT style, are visible.	WM_ERASEBKGD
@IDPAINT	Sent when a window created with @NOAUTODRAW needs repainting.	WM_PAINT
@IDMENUINIT	The @IDMENUINIT message is sent when a menu is about to become active. It occurs when the user clicks an item on the menu bar or presses a menu key. This allows the application to modify the menu before it is displayed.	WM_INITMENU
@IDTIMER	Sent when a timer, set with the STARTTIMER statement, expires. Multiple times may be used with a window. @CODE contains the timer ID.	WM_TIMER
@IDMENUMUPICK	Sent when a user selects a menu item. Check @MENUNUM for the ID of the menu item selected	WM_MENUSELECT
@IDCONTROL	Sent by controls when activated. Check @CONTROLID for the controls ID and	WM_COMMAND/ WM_NOTIFY

	@NOTIFYCODE for the notification message.	
@IDHSCROLL/ @IDVSCROLL	Sent when a horizontal or vertical scrollbar is used.	WM_HSCROLL/ WM_VSCROLL

See Also: [System Variables and Constants](#) for a list of predefined message ID's. You can of course handle any message that Windows sends to your handler. The compiler predefines the above constants as a convenience.

10.3 Printing Text in a Window

A blank window is uninteresting so lets put some text into it. Earlier we discussed how to use the PRINT statement to output text into the console window. To output text to a window we use the PRINT statement and specify the window as the first parameter.

In the console we could use LOCATE to specify where text would be printed. For a window we use the [MOVE](#) statement. Since both text and graphics are sent to a window as bitmapped images, the MOVE statement takes its parameters in pixels and not characters. Pixels start from the upper left corner at location 0,0. The syntax of MOVE is:

[MOVE](#) window, x, y

The window keeps track of the current position specified by the MOVE statement. When text is printed, the position is adjusted to the end of the line.

```

DEF w as WINDOW
OPENWINDOW w,0,0,640,200,@SIZE|@MINBOX|@MAXBOX,0,"Text",&main
CENTERWINDOW w
MOVE w,4,20
FOR x=0 TO 50
    PRINT w,"X"
NEXT x
MOVE w,4,40
PRINT w,"This is a test!"

run = 1
WAITUNTIL run = 0
CLOSEWINDOW w
END

SUB main( ),INT
SELECT @MESSAGE
    CASE @IDCLOSEWINDOW
        run = 0
    CASE @IDMOUSEMOVE
        MOVE w,4,60
        PRINT w,"Mouse Position: ",@MOUSEX,@MOUSEY," "
ENDSELECT
RETURN 0
ENDSUB

```

Changing the font

The text printed to a window will use the default font specified in the display control panel unless changed with the [SETFONT](#) statement. The SETFONT statement has the syntax of:

SETFONT window, typeface, height, weight {, flags | charset} {,ID}

Height and weight can both be 0 in which case a default size and weight will be used. Weight ranges from 0 to 1000 with 700 being standard for bold fonts and 400 for normal fonts. Flags can be a combination of *@SFITALIC*, *@SFUNDERLINE*, or *@SFSTRIKEOUT* for italicized, underlined, and strikeout fonts. If an ID is specified then the font of a control in the window or dialog is changed.

Example code fragment:

```
SETFONT mywin, "Ariel", 20, 700, @SFITALIC
PRINT mywin, "ARIEL bold italic"
```

Selecting character sets.

Certain fonts may have more than one character set. Normally this information is set automatically by the flag value returned by [FONTREQUEST](#). You can set the character set manually by using the following values ORed in with the flags

```
ANSI_CHARSET = 0
DEFAULT_CHARSET = 0x00010000
SYMBOL_CHARSET = 0x00020000
SHIFTJIS_CHARSET = 0x00800000
HANGEUL_CHARSET = 0x00810000
GB2312_CHARSET = 0x00860000
CHINESEBIG5_CHARSET = 0x00880000
OEM_CHARSET = 0x00FF0000
JOHAB_CHARSET = 0x00820000
HEBREW_CHARSET = 0x00B10000
ARABIC_CHARSET = 0x00B20000
GREEK_CHARSET = 0x00A10000
TURKISH_CHARSET = 0x00A20000
VIETNAMESE_CHARSET = 0x00A30000
THAI_CHARSET = 0x00DE0000
EASTEUROPE_CHARSET = 0x00EE0000
RUSSIAN_CHARSET = 0x00CC0000
MAC_CHARSET = 0x004D0000
BALTIC_CHARSET = 0x00BA0000
```

For example to set a terminal font which requires the OEM character set with an italic style:

```
SETFONT mywin, "Terminal", 20, 700, @SFITALIC | 0x00FF0000
```

If a character set doesn't exist in a particular font then the system will pick a font that closely matches the requested one.

Changing text colors

Text and graphics default to black on white. To change the current foreground drawing color of a window use the [FRONTPEN](#) statement. For the background color use the [BACKPEN](#) statement. The syntax of [FRONTPEN](#) and [BACKPEN](#) are:

[FRONTPEN](#) window, color

[BACKPEN](#) window, color

The color chosen by the [FRONTPEN](#) will be used by text, lines, outlines of rectangles and ellipses, and borders. The [BACKPEN](#) color is used as a fill for text if the drawing mode is not transparent.

The color variable can be set easily with the [RGB](#) function. [RGB](#) takes three numbers from 0 to 255 representing the intensity of red, green and blue components.

Example code fragments:

```
FRONTPEN mywin, RGB(0,0,255):REM light blue
FRONTPEN mywin, RGB(100,0,0):REM medium red
BACKPEN mywin,RGB(200,200,200):REM light gray
```

FONTREQUEST function

The [FONTREQUEST](#) function opens the standard system font dialog. The function returns the name of the font and sets four variables with the attributes of the requested font. The syntax of the [FONTREQUEST](#) function is:

name = [FONTREQUEST](#)(window, varSize, varWeight, varFlags, varColor {,dispname})

The variable parameters must be of type INT. [FONTREQUEST](#) returns an empty string if the user cancels the dialog.

Example code fragment:

```
DEF size,weight,flags,col:INT
DEF fontname:STRING

fontname = FONTREQUEST(win,size,weight,flags,col)
IF fontname <> ""
    SETFONT win,fontname,size,weight,flags
    FRONTPEN win,col
ENDIF
```

The variables can be preset to show initial font settings when the dialog is displayed. Optional *dispname* string presets the font name in the combobox of the system font dialog.

10.4 Graphics and Drawing

IWBASIC has graphic functions for lines, rectangles, ellipses, points, images, icons and cursors. Primitive graphic elements are drawn in the current foreground and background colors.

LINE and LINETO statements

The [LINE](#) statement draws a solid line between the start and end points specified. The [LINETO](#) statement draws a line between the last pen position and the end point specified. The line will be drawn in the current foreground color unless changed by the optional color parameter. The syntax of the line statements are:

[LINE](#) window startx, starty, endx, endy {, color}

[LINETO](#) window endx, endy {,color}

For the LINETO statement the last pen position is updated by any other graphic primitive or the MOVE statement

```
DEF w as WINDOW
OPENWINDOW w,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Lines",&main
CENTERWINDOW w
LINE w,4,20,620,20,RGB(255,0,0)
LINETO w,620,180,RGB(0,0,255)
LINETO w,4,180,RGB(0,255,0)
LINETO w,4,20,RGB(255,255,0)
run = 1

WAITUNTIL run = 0
CLOSEWINDOW w
END

SUB main( ),INT
SELECT @MESSAGE
CASE @IDCLOSEWINDOW
run = 0
ENDSELECT
RETURN 0
ENDSUB
```

RECT statement

The [RECT](#) statement is used to draw rectangles in the window. Rectangles can be filled with an optional fill color. The rectangle is drawn in the current foreground color unless a border color is specified. The syntax of the RECT statement is:

[RECT](#) window, left, top, width, height {, border {, fill} }

ELLIPSE statement

The [ELLIPSE](#) statement is used to draw ellipses in the window. The ellipse will be bound by the rectangle specified in the left,top,width and height parameters. The ellipse drawn in the current foreground color unless a border color is specified. The ellipse may be filled with an optional fill color. The syntax of the ellipse statement is:

ELLIPSE window, left, top, width, height {, border {, fill}}

CIRCLE statement

The **CIRCLE** statement is used to draw device independent circles. The circle will be drawn at the starting point specified with the radius specified. The circle will be drawn in the current foreground color unless a border color is specified. The circle may be filled with an optional fill color. The syntax of the **CIRCLE** statement is:

CIRCLE window, centerx, centery, radius {, border {, fill}}

Example:

```
DEF w as WINDOW
OPENWINDOW w,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Demo",&main
CENTERWINDOW w
RECT w,20,20,50,100,RGB(0,0,255),RGB(0,255,0)
ELLIPSE w,90,20,100,50,RGB(255,0,0),RGB(255,255,0)
CIRCLE w,250,75,50,RGB(0,255,0),RGB(0,0,255)
run = 1

WAITUNTIL run = 0
CLOSEWINDOW w
END

SUB main( ),INT
SELECT @MESSAGE
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN 0
ENDSUB
```

PSET statement

The **PSET** statement changes one pixel of the window to the foreground color or the optional specified color. The syntax of **PSET** is:

PSET window, x, y {, color}

GETPIXEL function

GETPIXEL retrieves the color of a pixel in the window at the coordinates specified. The syntax of **GETPIXEL** is:

color = **GETPIXEL**(window, x ,y)

Drawing Modes

The **DRAWMODE** statement sets the background mode. The background mode defines whether the system removes existing background colors on the drawing surface before drawing text. The syntax of the **DRAWMODE** statement is

DRAWMODE window, flags

Flags can be either `@TRANSPARENT` or `@OPAQUE`. If the mode is set to `@TRANSPARENT` then the background color is not changed when printing text.

Raster operations

The [RASTERMODE](#) statement sets the current drawing mode. The drawing mode specifies how the colors of the pen and the interior of filled objects are combined with the color already on the display surface. The syntax of RASTERMODE is:

[RASTERMODE](#) window, flags

Flags can be any one of the rastermode flags listed in the *appendix*.

Example:

```
DEF w as WINDOW
OPENWINDOW w, 0, 0, 640, 220, @SIZE | @MINBOX | @MAXBOX, 0, "Demo", &main
CENTERWINDOW w
RASTERMODE w, @RMXORPEN
RECT w, 20, 20, 50, 100, RGB(0, 0, 255), RGB(0, 255, 0)
ELLIPSE w, 20, 20, 100, 50, RGB(255, 0, 0), RGB(255, 255, 0)
CIRCLE w, 50, 50, 25, RGB(0, 255, 0), RGB(0, 0, 255)
run = 1

WAITUNTIL run = 0
CLOSEWINDOW w
END

SUB main( ), INT
SELECT @MESSAGE
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN 0
ENDSUB
```

Advanced graphics statements

For advanced users IW BASIC allows using WIN32 (WINAPI) functions to draw in a window. In order to use any of the WIN32 graphics functions you must first obtain a handle to a device context. While you could use the GetDC/ReleaseDC API functions, it is more convenient and compatible to use the built-in [GetHDC](#) and [ReleaseHDC](#) functions. The syntax of the GetHDC and ReleaseHDC functions are:

```
handle = GetHDC(window)
ReleaseHDC window, handle
```

The handle returned is an unsigned integer value and is a valid HDC. The advantages in using the built in functions is they are integrated with IW BASIC's auto drawing windows and will retain the current font, color and drawing mode settings.

The `@NOAUTODRAW` flag

To handle WM_PAINT messages directly in your program specify the @NOAUTODRAW flag when creating a window. Your windows handler subroutine will receive @IDPAINT messages whenever the window needs updating. A window created with @NOAUTODRAW uses less memory and will update faster. You must redraw the window every time in response to the @IDPAINT messages.

@NOAUTODRAW should also be used if your creating a window where another control covers the entire client area of the window. The *editor.iwb* sample uses this method to embed an edit control into a window.

Example code fragment:

```
' ...
OPENWINDOW w,0,0,350,350@SIZE|@NOAUTODRAW,0,"Test",&mainwindow
' ...
SUB mainwindow( ),INT
SELECT @MESSAGE
    CASE @IDCLOSEWINDOW
        run=0
    CASE @IDPAINT
        IF(bitmap)
            ShowImage w1,bitmap,0,x,y,w,h
        ENDIF
ENDSELECT
RETURN 0
ENDSUB
```

See Also: [Images, icons and cursors](#)

10.5 Images, Icons and Cursors

Loading an image

IWBASIC can load an image, icon or cursor with the [LOADIMAGE](#) function. LOADIMAGE returns a handle to the loaded image as an unsigned integer value. This value can then be passed to any of the functions that accepts a handle as a parameter. The syntax of LOADIMAGE is:

handle = [LOADIMAGE](#) (filename | resource ID, type)

Type is a numeric value defining what kind of image to load.

The valid values for type are:

@IMGBITMAP - bitmap (*.bmp)

@IMGICON - Icon (*.ico)

@IMGCURSOR - Cursor (*.cur)

@IMGEMF - Enhanced meta file (*.emf)

@IMGSCALABLE - scalable bitmap, JPEG (*.jpg) or GIF (*.gif) files.

@IMGOEM - OR in with @IMGBITMAP, @IMGICON or @IMGCURSOR to load an OEM (system) image.

@IMGMAPCOLORS - OR in with @IMGBITMAP to have the system search the bitmap color table and map the following shades of grey:

Dk Gray, RGB(128,128,128)	3DSHADOW COLOR
Gray, RGB(192,192,192)	3DFACE COLOR
Lt Gray, RGB(223,223,223)	3DLIGHT COLOR

The mapping of colors with @IMGMAPCOLORS is used in loading toolbars and button images so the current system colors are used for 3D elements.

If a filename is specified the image is loaded from disk. IWBASIC can also load bitmaps, icons and cursors directly from the executables resources. Enhanced meta files cannot be loaded from the resource table. Scalable images may be loaded from resources as custom type @RTIMAGE.

Resource ID is either the string or integer identifier of a resource compiled with the project.

Freeing the image

After your program is finished with an image it should call the DELETEIMAGE statement to free memory used by the image. The syntax of DELETEIMAGE is:

DELETEIMAGE handle, type

Type *must* be the same value specified in the LOADIMAGE function.

Displaying the image

An image can be drawn in a window using the SHOWIMAGE statement. SHOWIMAGE can draw images, icons and enhanced metafiles. The syntax of SHOWIMAGE is:

SHOWIMAGE window, handle, type, x, y, {w, h{, flags}}

Type is the same value specified in the LOADIMAGE statement. x and y specify the upper left corner of the image. W and h specify the width and height of the image or EMF file. The flags variable is for advanced users and are passed directly to the Windows function BitBlt.

If the image type equals 4 then SHOWIMAGE will show a JPEG, GIF or bitmap and w,h will scale the image to the width and height specified. If w and h are omitted the exact size of the image will be used with no scaling.

Width and Height must be specified for bitmap files.

Changing cursors

The currently displayed cursor in a window can be changed with the SETCURSOR statement. The syntax of SETCURSOR is:

SETCURSOR window, style {, handle}

Valid values for style are @CSWAIT for a wait cursor, @CSARROW for the standard arrow

cursor and `@CSCUSTOM` for a cursor loaded with the `LOADIMAGE` function.

Changing icons

The current icon for a window, displayed in the titlebar and taskbar, can be changed with the `SETICON` statement. `SETICON` accepts a handle from the `LOADIMAGE` function. The syntax of `SETICON` is:

`SETICON` window, handle

See Also: The samples `bitmap.iwb` and `imgview.iwb` for examples of using the image functions.

10.6 Creating and Using Menus

The `IWBASIC` compiler supports creating menus through both high level macro commands and low level API functions. The high level commands are an easy way to add an unlimited menu structure to your window or dialog. Context menus are also supported through the use of the high level menu creation macros. The macro statements closely follow the format used in menu resources to make converting between the two easier.

Adding a menu bar to a window or dialog

Begin creation of the menu bar with the `BEGINMENU` command. The window or dialog must be currently open.

```
BEGINMENU myWindow
```

Add menus to the menu bar with the `MENUTITLE` statement

```
MENUTITLE "&File"
```

Add items to the menu with the `MENUITEM` statement. The `MENUITEM` statement requires three parameters. The name of the item, any style flags and the ID of the menu item that will be used when your program receives menu messages. The ID cannot be 0.

```
MENUITEM "&Open", 0, 1
MENUITEM "Close", 0, 2
MENUITEM "Save", 0, 3
SEPARATOR
MENUITEM "Quit", 0, 4
ENDMENU
```

End the menu bar definition with the `ENDMENU` statement. Every `BEGINMENU` statement must be paired with a matching `ENDMENU` statement. The "&" in the item definitions specify the letter to be underlined in the item text. The item name string can contain any standard string escape sequences. Use the `SEPARATOR` statement to create the item separator line in the menu.

The style flags can a combination of [@MENUCHECK](#) for a menu that is initially checked and [@MENUDISABLE](#) for a menu that is initially disabled.

Creating submenus (popup menus)

Within a menu definition you can create unlimited levels of popup menus using the [BEGINPOPUP](#) and [ENDPOPUP](#) pair.

```
BEGINMENU myWindow
  MENUTITLE "&File"
    BEGINPOPUP "Load"
      MENUITEM "Text file", 0, 100
      MENUITEM "Word document", 0, 101
      MENUITEM "RTF File", 0, 102
    ENDPOPUP
  MENUITEM "Close", 0, 2
  MENUITEM "Save", 0, 3
  MENUITEM "Quit", 0, 4
  MENUTITLE "Edit"
    MENUITEM "Undo", 0, 255
    MENUITEM "Redo", 0, 256
ENDMENU
```

Inserting menus

To insert a new menu into an existing one use [BEGININSERTMENU](#) instead of [BEGINMENU](#). [BEGININSERTMENU](#) accepts two parameters, the window or dialog and the position to insert the new menu into. Position is zero based with 0 inserting before the first menu title, 1 inserting before the second, etc. [BEGININSERTMENU](#) must be used on MDI frame windows as there is already an existing menu containing the window tiling commands.

```
BEGININSERTMENU myWindow, 2
  MENUTITLE "Help"
    MENUITEM "About", 0, 300
ENDMENU
```

Creating context menus

Use the [CONTEXTMENU](#) statement in place of [BEGINMENU](#) to create and show a right-click context menu in your window or dialog. The x and y coordinates specified in the [CONTEXTMENU](#) statement are client coordinates. It is common to retrieve the coordinates from the [@MOUSEX](#) and [@MOUSEY](#) variables after an [@IDRBUTTONUP](#) message has been received.

```
'
SELECT @MESSAGE
CASE @IDRBUTTONUP
  CONTEXTMENU mywin,@MOUSEX,@MOUSEY
    MENUITEM "Color",0,99
```

```

MENUITEM "Clear",0,1
BEGINPOPUP "Line Size"
    MENUITEM "1", (linesize = 1) * @MENUCHECK,2
    MENUITEM "2", (linesize = 2) * @MENUCHECK,3
    MENUITEM "3", (linesize = 3) * @MENUCHECK,4
    MENUITEM "4", (linesize = 4) * @MENUCHECK,5
ENDPOPUP
ENDMENU

```

The `MENUTITLE` statement is not allowed in a context menu block as there is only one menu. It is important to remember that the context menu will be shown immediately and your message handler will receive menu messages from the context menu in the same manner as a normal menu.

Item addition and removal

Menu items can be added to an existing menu title by using the [ADDMENUITEM](#) command. The position specifies the zero based position of the menu to add the item to.

```
ADDMENUITEM myWindow, 3, "Register Online", 0, 75
```

Remove an existing menu or menu item by using the [REMOVEMENUITEM](#) command. The position specifies the zero based position of the menu or menu item to remove. If an ID of 0 is specified then the entire menu specified by position is removed

```
REMOVEMENUITEM myWindow, 3, 75
```

Menu appearance

Control the appearance of a menu or menu item using the [ENABLEMENU](#), [ENABLEMENUITEM](#) and [CHECKMENUITEM](#) statements. Click on the preceding links for descriptions.

```

'Disable the file menu
ENABLEMENU myWindow, 0, FALSE
'Check a menu item
CHECKMENUITEM myWindow, 4
'Disable a menu item
ENABLEMENUITEM myWindow, 255, FALSE

```

Handling menu messages

Once you define menus for your window, your programs handler subroutine will receive an `@IDMENUPICK` message whenever an item is selected. The ID of the menu will be returned in the `@MENUNUM` system variable.

Before the menu is displayed to the user an `@IDMENUINIT` message is sent to your handler to allow the program to make any modifications to the menu before it is shown. The modifications can include addition, removal, checking and disabling of menus and menu items.

Complete menu example:

```
REM define a window variable
```

```

DEF w1 as WINDOW
REM a variable to keep track of a checked menu
DEF bChecked as INT:bChecked = FALSE
REM open the window
OPENWINDOW w1,0,0,350,350,@MINBOX|@MAXBOX|@SIZE,0,"Simple Window",&main
REM define the menus
BEGINMENU w1
MENUTITLE "Option"
  MENUITEM "Print", 0, 1
  MENUITEM "Quit", 0, 2
  SEPARATOR
  MENUITEM "Check me", 0, 3
ENDMENU
REM print a message
PRINT w1,"Hello World "
REM when w1 = 0 the window has been closed
WAITUNTIL w1 = 0
END

SUB main( ),INT
SELECT @MESSAGE
  CASE @IDCLOSEWINDOW
    REM closes the window and sets w1 = 0
    CLOSEWINDOW w1
  CASE @IDMENUPIK
    SELECT @MENUNUM
      CASE 1: ' user selected Print
        PRINTWINDOW w1
      CASE 2: ' user selected Quit
        CLOSEWINDOW w1
      CASE 3: ' toggle a checkmark
        bChecked = (bChecked = FALSE)
        CHECKMENUITEM w1, 3, bChecked
    ENDSELECT
  ENDSELECT
RETURN 0
ENDSUB

```

Keyboard Accelerators

Keyboard accelerators, sometimes known as shortcut keys, can be added for any menu using the [ADDACCELERATOR](#) command. The format of the command is:

`ADDACCELERATOR window|dialog, fVirt, key, cmd`

fVirt is a flag describing whether the key needs to be combined with the SHIFT, ALT or CTRL key in order to activate the accelerator. *Key* can be an ASCII key code or one of the virtual key codes listed in the appendix. *Cmd* is the menu ID.

Example:

```

BEGINMENU win
  MENUTITLE "&File"
  MENUITEM "Load File\tCtrl+L",0,1
  MENUITEM "Save\tCtrl+S",0,2
  MENUITEM "Print\tAlt+P",0,4
  MENUITEM "Quit\tCtrl+C",0,3
  MENUTITLE "&Options"

```

```
MENUITEM "Change Font\tF4",0,5
ENDMENU

'add our keyboard accelerators
ADDACCELERATOR win,@FCONTROL|@FVIRTKEY,ASC("L"),1
ADDACCELERATOR win,@FCONTROL|@FVIRTKEY,ASC("S"),2
ADDACCELERATOR win,@FCONTROL|@FVIRTKEY,ASC("C"),3
ADDACCELERATOR win,@FALT|@FVIRTKEY,ASC("P"),4
ADDACCELERATOR win,@FVIRTKEY,0x73,5:'F4 key changes font
```

Low level API menu functions

The low level functions are part of the standard command set and are for advanced menu creation. They are used by the higher level macros to actually create and attach the menus to the window or dialog. The low level functions are equivalents to the Windows API functions for manipulating menus and only differ in that they understand IWBASIC WINDOW and DIALOG variables and check for NULL handle values.

In order to construct a menu bar you first need to call [CreateMenu](#) with no parameters.

```
hMenu = CreateMenu( )
```

The handle returned is of type UINT and is a standard Windows HMENU. Once the main menu is created you construct the various levels by using CreateMenu with the parameter set to 1 to create the dropdown menu titles or embedded popup menus. Attach the popup menu to the menu bar using [APPENDMENU](#).

```
hMenu = CreateMenu()
hPopup = CreateMenu(1)
APPENDMENU(hMenu, "File", MF_POPUP|MF_STRING, hPopup)
```

Add menu items to the newly created popup menus by using APPENDMENU

```
APPENDMENU(hPopup, "Open", MF_STRING, 1)
APPENDMENU(hPopup, "Quit", MF_STRING, 2)
```

And finally replace the menu bar in the window or dialog with your newly created one by calling [SETMENU](#)

```
SETMENU win, hMenu
```

See any Windows API guide for more complex examples on using menu creation functions. Constants for APPENDMENU can be found in the API Viewer or on MSDN.

10.7 Creating Embedded Browsers

IWBASIC allows creating a self contained internet browser attached to a window. The embedded browser functions require Internet Explorer 4.0 or greater to be installed on the system. To embed a browser open a window and use the [ATTACHBROWSER](#) command. @NOAUTODRAW

should be used as a window style to prevent any overwriting of the displayed HTML document. ATTACHBROWSER returns -1 if the browser could not be created or 0 if it was successfully attached to the window.

Example:

```
DEF wb as WINDOW
OPENWINDOW wb,0,0,640,480,@SIZE|@NOAUTODRAW|@MINBOX|@MAXBOX,0,"Test Browse",&main

IF ATTACHBROWSER(wb,"http://www.ionicwind.com") = -1
    MESSAGEBOX wb, "Couldn't create browser control","Error"
    END
ENDIF

run = 1
WAITUNTIL run=0
CLOSEWINDOW wb
END

SUB main( ),INT
SELECT @MESSAGE
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN 0
ENDSUB
```

The browser automatically adjusts its size to the containing window. No extra action is required.

Controlling the browser

As indicated in the above example, controlling the embedded browser is done with [BROWSECMD](#). BROWSECMD serves as both a statement and a function depending on the command issued. The syntax of BROWSECMD is:

```
{return = } BROWSECMD (window, command {,parameters} )
```

The available commands are:

Command	Details
@NAVIGATE, strURL	Navigates to the specified web page or local file.
@GOHOME	Loads the default 'home' page
@GOBACK	Moves back one page in the history list.
@GOFORWARD	Moves ahead one page in the history list.
@BROWSESEARCH	Navigates to the search page specified in the users internet options.
@BROWSESTOP	Stops loading of the current document.

@REFRESH	Refreshes the current document.
@GETTITLE, strTitle {,cchTitle}	
@BACKENABLED	Returns 1 if there is at least one page to @GOBACK to or 0 otherwise.
@FORWARDENABLED	Returns 1 if there is at least one page to @GOFORWARD to or 0 otherwise.
@CANCELNAV	Cancels navigation to the current page. Use in response to @IDBEFORENAV to limit navigation.
@GETPOSTDATA, pstrData {,cchData}	Available when @IDBEFORENAV is sent.
@GETHEADERS, pstrHeaders {,cchHead}	Available when @IDBEFORENAV is sent.
@GETNAVURL, pstrURL {,cchURL}	Available when @IDBEFORENAV or @IDNAVCOMPLETE is sent.
@BROWSELOAD, strHtml	Loads the browser with the contents of a string.
@BROWSEPRINT	Prints the currently displayed document. No Parameters.
@GETSTATUSTEXT, pstrStatus {,cchStatus}	Gets the status text to display in a status bar. Available when the @IDSTATUSTEXTUPDATE message is sent

The commands that return string data do so through the string parameter supplied. The optional parameter is always the length of the supplied string and defaults to 255, the length of a standard STRING type. For example getting the status text might look like this:

```
DEF strStatus[500] as ISTRING
'
BROWSECMD wb, @GETSTATUSTEXT, strStatus, 500
```

For readability we will include the length parameter in our examples even when using standard strings.

Navigating

The @NAVIGATE command allows specifying either a network URL or path to a local file or directory. When used with a directory path the browser will display an Explorer like window allowing standard file operations. The embedded browser can display any of the file types that are viewable in Internet Explorer including pictures, plug-ins, and html documents.

Example navigation statements:

```
BROWSECMD window, @NAVIGATE, "c:\""
BROWSECMD window, @NAVIGATE, "e:\\images\\picture.jpg"
BROWSECMD window, @NAVIGATE, "http://www.ionicwind.com"
BROWSECMD window, @NAVIGATE, "ftp://ftp.ionicwind.com"
```

Messages

The embedded browser will send an `@IDBEFORENAV` message to the window just before it navigates to a page. If you wish to limit access to certain websites or files you can call `BROWSECMD` with a command of `@CANCELNAV`. This message can also be used to extract the data from an HTML form.

Once navigation is complete the browser will send an `@IDNAVCOMPLETE` message. Use this message to redirect to a different URL or to save data gathered during an `@IDBEFORENAV` message.

During operation of the browser control your handler will also receive `@IDSTATUSTEXTUPDATE` messages. Use the `BROWSECMD @GETSTATUSTEXT` to retrieve the text to display. For example when hovering over links the status text sent would be the URL of the link.

An example of a browser control used to collect form data:

```
DEF wb as WINDOW
DEF data,url as STRING
OPENWINDOW wb,0,0,640,480,@SIZE|@NOAUTODRAW|@MINBOX|@MAXBOX,0,"Test Browse",&main

ATTACHBROWSER wb

BROWSECMD wb,@NAVIGATE,"c:\\forms\\userdata.html"

run = 1
WAITUNTIL run=0
CLOSEWINDOW wb
END

SUB main( ),INT
SELECT @MESSAGE
CASE @IDCLOSEWINDOW
    run = 0
CASE @IDBEFORENAV
    BROWSECMD(wb,@GETNAVURL, url, 255)
    IF url <> "c:\\forms\\userdata.html"
        BROWSECMD(wb,@GETPOSTDATA, data, 255)
    ENDIF
CASE @IDNAVCOMPLETE
    BROWSECMD(wb,@GETNAVURL, url, 255)
    IF url <> "c:\\forms\\complete.html"
        BROWSECMD wb,@NAVIGATE,"c:\\forms\\complete.html"
    ENDIF
ENDSELECT
RETURN 0
ENDSUB
```

In the above example we first navigate to an html page containing a form. In response to `@IDBEFORENAV` we check the URL to see if it is not the first page we navigated to. This means the user has pressed the 'send' or 'post' button. Then we retrieve the posted data. In a real world example we would parse the data in the string and act upon it. If one of the fields was in error then the `@CANCELNAV` command could prevent the user from continuing on.

Posted data normally consists of name-value pairs separated by the '&' symbol. If our form had two fields 'first' and 'last' and the user entered John Smith the string returned would be:

```
first=John&last=Smith
```

Notes

Be careful when navigating to a page in response to @IDNAVCOMPLETE. If you do not test the current URL, the browser will end up in a loop since every page sends the message.

Don't use the @NAVIGATE command in response to an @IDBEFORENAV message. Doing so will send the browser into an endless loop and end your program without warning.

See Also: The *browser_test.iwb* sample program for a fully functional web browser based on the embedded browser control.

10.8 Using Dialogs

Creating a Dialog

Using dialogs in IW BASIC is similar to defining and opening a window. Many of the same window statements and functions apply to dialogs as well. Dialogs are first defined using the [CREATEDIALOG](#) statement and then shown elsewhere in your program with the [DOMODAL](#) or [SHOWDIALOG](#) commands. The syntax of the CREATEDIALOG statement is:

```
CREATEDIALOG variable, Left, Top, Width, Height, flags, parent, title, procedure
```

The variable must have been previously defined as type DIALOG with the DEF / DIM statement. *Procedure* refers to the address of the subroutine that will handle messages from this dialog, use the & operator to specify the address. *Flags* should contain @CAPTION for a title and @SYSTEMMENU for the standard system menu and close button. If @CAPTION is omitted the dialog will not have a title bar and will be fixed in place.

It is important to note the difference between a dialog and a window. A window is created and shown with one statement, OPENWINDOW. A dialog is *defined* by the CREATEDIALOG statement but not shown until you use DOMODAL or SHOWDIALOG. This means a dialog is reusable and can be shown many times without having to recreate it each time.

The *parent* parameter specifies the owner window/dialog for this dialog. If used the CREATEDIALOG statement must appear after the parent is opened. To alleviate this restriction specify the parent window/dialog in the DOMODAL or SHOWDIALOG statements.

The [Form Editor](#) provides an easy interface for designing dialogs and placing controls.

Showing the dialog

Add controls to the dialog after it is defined and show the dialog with the `DOMODAL` or `SHOWDIALOG` functions.

`DOMODAL` shows the dialog as modal. This means that all other windows in your program will be blocked until the dialog is closed. The `DOMODAL` function has the syntax of:

```
return = DOMODAL( variable [,parent] )
```

The variable must be of type `DIALOG` and defined with the `CREATEDIALOG` statement. `DOMODAL` will return the value given to the `CLOSEDIALOG` statement or `@IDCANCEL` if the user presses the <Esc> key to dismiss the dialog. All input will be captured by the dialog while it is displayed. Note that any controls in the dialog cannot be initialized until the dialog is displayed. All control initialization should be done in the dialog handler subroutine in response to the `@IDINITDIALOG` message.

The optional parent parameter overrides the parent window/dialog specified in the `CREATEDIALOG` statement. It is preferable to specify the parent window/dialog when it is shown.

To show a non modal dialog use the `SHOWDIALOG` statement. The syntax of `SHOWDIALOG` is:

```
SHOWDIALOG variable [,parent]
```

Once the dialog is displayed, your program continues to execute normally. A dialog shown with `SHOWDIALOG` requires a message loop to properly process and send message to the handler subroutine. In this respect a dialog shown with `SHOWDIALOG` operates in the same manner a normal window does with the benefits of a dialog.

The optional parent parameter overrides the parent window/dialog specified in the `CREATEDIALOG` statement. It is preferable to specify the parent window/dialog when it is shown.

Closing the dialog:

```
CLOSEDIALOG variable, return_value
```

The *return_value* can be any integer value, it is ignored for non modal dialogs shown with the `SHOWDIALOG` statement. You can use the predefined values of `@IDOK` and `@IDCANCEL` if applicable.

When your dialog is about to be displayed, the handler for the dialog will receive the message `@IDINITDIALOG`. This is a good place to perform any initializations such as centering the dialog with the `CENTERWINDOW` statement and presetting any controls.

You should copy any control data before the dialog is closed. After a dialog is closed all of the controls are invalid and accessing them will fail.

Example:

```

DEF dl:DIALOG
DEF w:WINDOW
DEF result:INT
DEF answer:STRING
'Open our window and define a dialog
OPENWINDOW w,0,0,640,200,@SIZE,0,"Dialog Test",&wndproc
CREATEDIALOG dl,0,0,100,100,@CAPTION|@SYSMENU,w,"My Dialog",&dialoghandler
CONTROL dl,@BUTTON,"OK",25,75,50,20,@TABSTOP,1
CONTROL dl,@EDIT,"",15,45,70,14,@TABSTOP,10
'Show the dialog
result = DOMODAL dl
'Print the result
MOVE w,4,20
IF result = @IDOK
    PRINT w, answer
ELSE
    PRINT w, "DIALOG canceled"
ENDIF
'Just wait for the window to be closed
run=1
WAITUNTIL run = 0
CLOSEWINDOW w
END

'Our window subroutine
SUB wndproc( ),INT
SELECT @MESSAGE
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN 0
ENDSUB

'Our dialog subroutine
SUB dialoghandler( ),INT
SELECT @MESSAGE
    CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 1
                answer = GETCONTROLTEXT(dl, 10)
                CLOSEDIALOG dl,@IDOK
            ENDSELECT
        'All controls should be initialized while processing the @IDINITDIALOG message
        CASE @IDINITDIALOG
            CENTERWINDOW dl
            SETCONTROLTEXT dl,10,"Yipee!"
    ENDSELECT
RETURN 0
ENDSUB

```

The dialog can be created without a parent window in which case your program would be a dialog application.

10.9 MDI Windows

Multiple Document Interface, or MDI, windows consist of a parent frame window and one or

more child windows. Most word processors use the MDI interface as well as IWBASIC IDE. To create a MDI interface first open a window with @MDIFRAME as one of the style flags:

```
OPENWINDOW frame,0,0,640,400,@MDIFRAME|@SIZE,0,"Main",&main
```

Any child window that uses an MDI frame as the parent parameter becomes an MDI child window:

```
OPENWINDOW w,0,0,200,100,@SIZE|@MINBOX|@MAXBOX,frame,"Child",&main
```

The frame window will automatically contain a standard menu for minimizing, maximizing, restoring and arranging icons of the child windows. When ending your program it is not necessary to close all of the child windows individually. Closing just the frame window will also close all of the child windows.

For default sized child windows you can use a special variable @USEDEFAULT for the *left* parameter. Windows will then pick a standard size for your window based on the current client size of the frame window:

```
OPENWINDOW w,@USEDEFAULT,0,0,0,@SIZE|@MINBOX|@MAXBOX,frame,"Child",&main
```

You can use the same message subroutine for both the frame window and the child windows. In order to differentiate where the message is coming from, use the special @HITWINDOW variable. @HITWINDOW will contain a pointer to the originating window when your message subroutine is called. The following example fragment demonstrates this:

```
SUB main( ),INT
SELECT @MESSAGE
CASE @IDCLOSEWINDOW
IF #<WINDOW>@HITWINDOW = frame
run = 0
ELSE
CLOSEWINDOW #<WINDOW>@HITWINDOW
ENDIF
ENDSELECT
RETURN 0
ENDSUB
```

Notes:

The client of the frame window will automatically adjust its size to accommodate status windows and toolbars. You can't draw or PRINT into the frame window as it is just a place holder for MDI child window. The color of the client area in a frame window is determined by the users control panel settings.

The example *mdidemo.iwb* contains a complete example of a skeleton MDI program. It is a good place to start building MDI applications.

10.10 Information Functions

General functions to query a window, dialog or control properties.

Retrieving sizes

GETSIZE window | dialog, varL, varT, varW, varH {, ID}

Will return the size of a window, dialog or control in the variables specified. Variables must be of type INT. The left and top coordinates are relative to the upper left corner of the screen. If *ID* is specified then the size of the control is returned.

See Also: [SETSIZE](#)

GETCLIENTSIZE window | dialog, varL, varT, varW, varH

Will return the size of the drawing area of the window in the variables specified. *varL* and *varT* will always be 0 with this function. Variables must be of type INT.

GETSCREENSIZE varW, varH

Returns the system screen size. Useful for determining a size to set your window. Variables must be of type INT.

GETTEXTSIZE window, string, varWidth, varHeight

Returns the size of a string in pixels when printed to the window. *varWidth* and *varHeight* must be of type INT. The size of the string is useful for setting line positions with the MOVE statement.

Retrieving drawing and caret positions

GETPOSITION window, varX, varY

Retrieves the current drawing position in the window. *varX* and *varY* must be of type INT. The current drawing position is set with the [MOVE](#) statement, by certain graphics operations, and by the [PRINT](#) statement.

GETCARETPOSITION window, varX, varY

Retrieves the current caret position in the window. *varX* and *varY* must be of type INT. The *window* parameter is maintained for backwards compatibility with older versions of IW BASIC; It is not used by the function and can be passed an uninitialized WINDOW variable.

Miscellaneous

caption\$ = **GETCAPTION** (window | dialog)

Returns the windows caption text. Use [GETCONTROLTEXT](#) to retrieve the text of button controls.

See also: [SETCAPTION](#), [SETCONTROLTEXT](#)

DC = **GETHDC**(window)

Safe way to access a device context to an IW BASIC created window. The returned dc can be

used with any Windows API function that requires a device context to the window.

See also: [RELEASEHDC](#)

10.11 ScrollBars

[SETSCROLLRANGE](#) window | dialog, ID, min, max

[SETSCROLLRANGE](#) Sets the minimum and maximum range of a scrollbar control to min and max. All values returned by the scrollbar will be between min and max. If ID = -1 then sets the range of the windows horizontal scrollbar. If ID = -2 then sets the range of the windows vertical scrollbar. ID must be a scrollbar control.

[GETSCROLLRANGE](#) window | dialog, ID, varMin, varMax

Stores the scrollbars range into the variables specified by varMin and varMax. The variables must be of type INT. If ID = -1 then stores the range of the windows horizontal scrollbar. If ID = -2 then stores the range of the windows vertical scrollbar. ID must be a scrollbar control. :

[SETSCROLLPOS](#) window | dialog, ID, position

Sets the slider position of a scrollbar. If ID = -1 then sets the scroll position of the windows horizontal scrollbar. If ID = -2 then sets the scroll position of the windows vertical scrollbar. Any other ID value is a user-defined scrollbar. Position must be between the minimum and maximum values set by the [SETSCROLLRANGE](#) statement.

position = [GETSCROLLPOS](#) (window | dialog, ID)

Returns the slider position of a scrollbar. If ID = -1 then returns the scroll position of the windows horizontal scrollbar. If ID = -2 then returns the scroll position of the windows vertical scrollbar. Any other ID value is a user-defined scrollbar.

position = [GETTHUMBPOS](#)(window | dialog, ID)

Returns the current thumb track position of a scrollbar. If ID = -1 then returns the thumb track position of the windows horizontal scrollbar. If ID = -2 then returns the thumb track position of the windows vertical scrollbar. Any other ID value is a user-defined scrollbar.

10.12 Controls

10.12.1 Control Creation

Introduction

IW BASIC supports dynamic creation of controls in windows or dialogs. In a window the control is created and added immediately. In a dialog the control is added to the dialog template and created when the dialog is shown.

The [CONTROL](#) statement is used for both windows and dialogs. The generalized syntax of the control statement is:

CONTROL parent, type, title, left, top, width, height, style_flags, id

The parent of the control must be specified and can be either a WINDOW or DIALOG variable. The dimensions of the control are in pixels and are automatically converted to device units for dialogs. This allows the same control definition to be used for both a dialog and a window.

In order to use the [CONTROL](#) command the type of the control must be one of the following control constants:

```
@BUTTON
@CHECKBOX
@RADIOBUTTON
@EDIT
@LISTBOX
@COMBOBOX
@STATIC
@SCROLLBAR
@GROUPBOX
@RICEDIT
@LISTVIEW
@STATUS
@SYSBUTTON
@RGNBUTTON
@TREEVIEW
```

Example definitions:

```
CONTROL w,@BUTTON,"Save",56,100,50,20,0, 1
CONTROL d,@COMBOBOX,"",450,100,100,100,@TABSTOP, 7
```

Each control has its own creation flags and returns specific messages to your window or dialog subroutine, see the specific topics for each control type for details. The @IDCONTROL message is sent when any operation is done to one of the controls. @CONTROLID will contain the ID value specified when the control was created. @WPARAM may contain additional information. Also it is important to check the value of @NOTIFYCODE for most control types. Communication with the control is done with generic control functions and the [CONTROLCMD](#) function. They will be covered in detail for each individual control type.

The easiest way to create and size a control is with the [Fom Editor](#).

Creating other types of controls

Windows supports a wide variety of built in and third party controls. To create a control not

directly supported by the [CONTROL](#) statement and [Form Editor](#) use the [CONTROLEX](#) statement and specify the class name of the control. The syntax of the CONTROLEX statement is:

CONTROLEX parent, class, title, left, top, width, height, style, exStyle, ID

The parent of the control must be specified and can be either a WINDOW or DIALOG variable. The dimensions of the control are in pixels and are automatically converted to device units for dialogs. This allows the same control definition to be used for both a dialog and a window. All control styles must be explicitly specified since the compiler can't predict what visual styles are common for the control. The only control styles automatically added are WS_VISIBLE and WS_CHILD. *exStyle* is the extended window style of the control. The following extended styles are predefined for use with CONTROLEX:

@EXCLIEN TEDGE - Specifies that a control has a 3D look — that is, a border with a sunken edge.

@EXSTATICEDGE - Creates a control with a three-dimensional border style intended to be used for items that do not accept user input.

@EXWINDOWEDGE - Specifies that a control has a border with a raised edge.

@EXLEFT - Gives a control generic left-aligned properties. Default.

@EXRIGHT - Gives a control generic right-aligned properties. This depends on the control class.

After the control is created you can use the basic control manipulation functions on it such as SETFONT, SETCONTROLTEXT, SETSIZE, GETSIZE, etc. Other functionality of the specific control can be achieved by using [SENDMESSAGE](#). The custom control is automatically destroyed when the window or dialog is closed so there is no need to use the DestroyWindow API function. The common control library is also automatically initialized when creating the control.

The CONTROLEX statement will return the window handle of the created control when used with a window parent. If used with a dialog the handle to the control can be obtained with the [GETCONTROLHANDLE](#) function during the @IDINITDIALOG message.

Example creating a progress bar control:

```
CONST PBM_SETPOS = 0x402
CONST PBM_SETRANGE32 = 0x406
CONST PBM_SETSTEP = 0x404
CONST PBM_STEPIT = 0x405

DIALOG d1
CREATEDIALOG d1,0,0,250,100,@CAPTION|@SYSTEMMENU|@BORDER,0,"Dialog App",&dialoghandler
CONTROLEX d1,"msctls_progress32","",4,40,236,20,@BORDER,@EXCLIEN TEDGE,20

DOMODAL d1
END

SUB dialoghandler( ),INT
SELECT @MESSAGE
CASE @IDINITDIALOG
CENTERWINDOW d1
'set the progress bars range and position
SENDMESSAGE d1,PBM_SETRANGE32,0,500,20
SENDMESSAGE d1,PBM_SETPOS,250,0,20
```

```
CASE @IDCLOSEWINDOW
    CLOSEDIALOG d1,@IDOK
ENDSELECT
RETURN 0
ENDSUB
```

Class names for the Windows common controls:

Header Control - "SysHeader32" **
Toolbar Control - "ToolbarWindow32" **
ReBar Control - "ReBarWindow32" **
ToolTips Control - "tooltips_class32" **
TrackBar Control - "msctls_trackbar32" **
UpDown Control - "msctls_updown32" **
Progress Control - "msctls_progress32" **
HotKey Control - "msctls_hotkey32"
ComboBoxEx - "ComboBoxEx32" **
Tab Control - "SysTabControl32" **
Animate Control - "SysAnimate32"
Month Calendar Control - "SysMonthCal32" **
Date/Time Control - "SysDateTimePick32" **
IP Address Edit - "SysIPAddress32" **
Pager Control - "SysPager" **
NativeFont Control - "NativeFontCtl"

The above list is not all inclusive as Microsoft adds new common control types with each release of Windows. The usage and messages the controls use can be found by searching Google or MSDN

** Note: These controls are currently implemented via custom wrappers instead of using [CONTROL](#) or [CONTROLEX](#). Each is covered in its own sub-section of the controls section.

10.12.2 General Control Functions - WIP

The functions and commands outlined here are for communicating with and initializing various control types. See the individual sections on each control for other possible operations and commands.

When used with controls in a dialog use SETCONTROLCOLOR in response to the @IDINITDIALOG message. The foreground color is used by text displayed in the controls. These colors can be specified using the RGB function.

@BN_CLICKED
@BN_DBLCLK
@LEFTTEXT

@DISABLE
@SS_SIMPLE
@SS_LEFT
@SS_CENTER
@SS_RIGHT
@SS_NOTIFY

@BN_SETFOCUS
@BN_KILLFOCUS
@BS_BOTTOM
@BS_CENTER
@BS_LEFT
@BS_NOTIFY
@BS_RIGHT
@BS_TOP
@BS_VCENTER

10.12.3 Button Controls

A *button* is a control the user can click to provide input to an application. IWBasic provides three types of buttons.

They are covered in the following sub-sections:

- [@Button Controls](#)
- [@RgnButton Controls](#)
- [@SysButton Controls](#)

10.12.3.1 @Button Controls

About Button controls

A Button is a control the user can click to provide input to an application. Button controls have the capability of displaying a bitmap instead of text.

Creating the control

@Button controls are created either through the [Form Editor](#) or manually with the [CONTROL](#) statement. The following statement (with no flags applied)

```
CONTROL w1,@BUTTON,"Button 1",27,56,100,25,0,w1_CLRBUTTON1
```

results in this



Button control styles

The following `@BUTTON` style flags can be specified in the `CONTROL` statement or by ticking the corresponding check box in the control properties dialog of the [Form Editor](#):

`@CTLBTNDEFAULT`

Identifies this as the default push button in a dialog

`@CTLBTNBITMAP`

Defines a bitmap button. To specify the bitmap to display set the button's text to the complete pathname to the bitmap file with the `SETCONTROLTEXT` statement. If the button is contained in a dialog use `SETCONTROLTEXT` in response to the dialog's `@IDINITDIALOG` message.

Example of bitmap `@BUTTON` control:

```
DEF d1 as DIALOG

CREATEDIALOG d1,0,0,295,168,0x80C80080, 0, "Bitmap Test", &dialog_main
CONTROL d1,@BUTTON,"",27,56,100,25,@CTLSTCBITMAP, 1

DOMODAL d1
END

SUB dialog_main( ),INT
SELECT @MESSAGE
CASE @IDINITDIALOG
SETCONTROLTEXT d1,1,GETSTARTPATH + "bug.bmp"
CENTERWINDOW d1
ENDSELECT
RETURN 0
ENDSUB
```

The above code results in a button that looks like this



`@CTLBTNFLAT`

Creates a flat button. This code

```
CONTROL w1,@BUTTON,"Button 1 ",27,56,100,25,@CTLBTNFLAT,w1_CLRBUTTON1
```

results in

Button 1

`@CTLBTNMULTI`

Creates a multi line button control. Text is automatically word wrapped. This code

```
CONTROL w1,@BUTTON,"Button with two lines ",27,56,100,40,@CTLBTNMULTI,w1_CLRBUTTON1
```

results in

Button with
two lines

`@DISABLE`

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are

grayed out and cannot be selected. This code

```
CONTROL w1,@BUTTON,"Button 1 ",27,56,100,25,@DISABLE,w1_CLRBUTTON1
```

results in . A control disabled in this manner can be enabled via the [ENABLECONTROL](#) command.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSTEMMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

@Button control functions and statements

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

GETCONTROLTEXT is used to retrieve the text of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control. The dimensions returned include the borders.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wParam, lParam, ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the text of a control. If the control is a bitmap button then the text is the path to a bitmap file.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight, flags, ID

Changes the font of all text in the control.

[SETSIZE](#) window | dialog, L, T, W, H, ID

Changes the size of the control. The dimensions include the borders of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

Notification messages

These are the messages sent from the control to the control's parent window/dialog.

@BN_CLICKED

The control has been clicked.

@BN_DBLCLK

The control has been double clicked.

10.12.3.2 @RgnButton Controls

About @RgnButton controls

A RgnButton (region button) is a control the user can click to provide input to an application. The RgnButton is an extension of the @BUTTON control that allows using regions to define non-rectangular buttons. Region buttons support automatic hot tracking using either a solid color or bitmap. Unlike the @BUTTON type, the region buttons use a normal text caption and support font and border changes when using bitmaps. They are backwards compatible with the @BUTTON type and can be used as a direct replacement without breaking existing code.

Creating the control

RgnButton controls are created either through the [Form Editor](#) or manually with the [CONTROL](#) statement. The following statement (with no flags applied)

```
CONTROL w1,@RGNBUTTON,"RgnBtn 1",26,96,100,25,0,w1_RGNBUTTON1
```

results in this  which is the basic button and is backward compatible to the @BUTTON control.

@RgnButton control styles

The following @RgnButton style flags can be specified in the [CONTROL](#) statement or by ticking the corresponding check box in the control properties dialog of the [Form Editor](#):

@CTLBTNDEFAULT

Identifies this as the default push button in a dialog

@CTLBTNBITMAP

Defines a bitmap button. To specify the bitmap to display set the button's text to the complete pathname to the bitmap file with the [SETCONTROLTEXT](#) statement. If the button is contained in a dialog use [SETCONTROLTEXT](#) in response to the dialog's @IDINITDIALOG message.

Example of bitmap @RGNBUTTON control:

```
DEF d1 as DIALOG

CREATEDIALOG d1,0,0,295,168,0x80C80080, 0, "Bitmap Test", &dialog_main
CONTROL d1,@RGNBUTTON,"",27,56,100,25,@CTLSTCBITMAP, 1

DOMODAL d1
END

SUB dialog_main( ),INT
SELECT @MESSAGE
CASE @IDINITDIALOG
SETCONTROLTEXT d1,1,GETSTARTPATH + "bug.bmp"
CENTERWINDOW d1
ENDSELECT
RETURN 0
ENDSUB
```

The above code results in a button that looks like this



@CTLBTNFLAT

Creates a flat button. This code

```
CONTROL w1,@RGNBUTTON,"Button 1 ",27,56,100,25,@CTLBTNFLAT,w1_RGNBUTTON1
```

results in

RgnBtn 1

@CTLBTNMULTI

Creates a multi line button control. Text is automatically word wrapped. This code

```
CONTROL w1,@RGNBUTTON,"Button with two lines ",27,56,100,40,@CTLBTNMULTI,w1_RGNBUTTON1
```

results in

**RgnBtn with
two lines**

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected. This code

```
CONTROL w1,@RGNBUTTON,"Button 1 ",27,56,100,25,@DISABLE,w1_RGNBUTTON1
```

results in

RgnBtn 1

. A control disabled in this manner can be enabled via the [ENABLECONTROL](#) command.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSTEMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

@RgnButton control functions and statements

hRgn = [RGNFROMBITMAP](#) (id, transparent color, tolerance)

Creates a window region from a bitmap. The command allows easy creation of regions from a bitmap by specifying a transparency color. The region handle returned is a standard HRGN and can be used with either the region buttons or to create non-rectangular windows using the WinAPI

function `SetWindowRegion`. A region bitmap can be created using any drawing editor like MSPaint. The region bitmap is a two color mask. The user can use any two colors. Everywhere one color is located will be transparent and appear to not exist on the final control. The other color will appear as "inside" the control boundary and allow for the displaying of text and other bitmaps. Using black and white with black as the transparent color makes it easier to visualize during development. The rgn bitmap needs to have the same dimensions as the basic rgnbutton created with the

`CONTROL` command. The following is an example of a rgn bitmap:



[SETBUTTONRGN](#) (window | dialog, ID, hRgn)

Sets the display region used by a button of type. After this command is executed the button will now take on the shape of the visible color of the rgn bitmap. This code:

```
CONTROL w1,@RGNBUTTON,"RgnBtn 1",26,96,87,30,0,w1_RGNBUTTON1
SETBUTTONRGN w1,w1_RGNBUTTON1,RGNFROMBITMAP(GETSTARTPATH+"rgn_bmp2.bmp")
```

results in this



`hRgn = COPYRGN` (hRgn)

Copies a Windows region. This command is used when it is desired to use the same rgn bitmap for multiple buttons. For example, if we wanted 4 buttons to all look like the diamond shape above the region creating code would look like this:

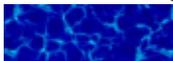
```
uint hRgn = RGNFROMBITMAP(GETSTARTPATH+"rgn_bmp2.bmp")
uint hRgn2 = COPYRGN(hRgn)
uint hRgn3 = COPYRGN(hRgn)
uint hRgn4 = COPYRGN(hRgn)
```

[DELETERGN](#) (hRgn)

Deletes a region. If a rgn is created with either [RGNFROMBITMAP](#) or [COPYRGN](#) and is never assigned to a rgnbutton then this command is used to free up memory. If the region is assigned to a rgnbutton then the memory is automatically freed when the rgnbutton is destroyed.

[SETBUTTONBITMAPS](#) (window | dialog, ID, hNormal, hHot, hSelected)

Sets the bitmaps used by the button. For example, Let's say we want to use this image



for the normal image and this



for the hot image ('selected' is for future

use and should be set to null). The following code

```
SETCONTROLCOLOR w1,w1_RGNBUTTON1,RGB(0,255,0),0
SETBUTTONRGN w1,w1_RGNBUTTON1,RGNFROMBITMAP(GETSTARTPATH+"rgn_bmp2.bmp")
SETBUTTONBITMAPS w1,w1_RGNBUTTON1, _
    LOADIMAGE(GETSTARTPATH+"button_bmp_normal2.bmp",@IMGBITMAP),_
    LOADIMAGE(GETSTARTPATH+"button_bmp_normal.bmp",@IMGBITMAP),_
0
```

results is this  when the mouse is not over the button and this  when the mouse is over the button. Notice that text color is set with with the [SETCONTROLCOLOR](#). If the button uses bitmaps the background is not used.

[SETBUTTONBORDER](#) (window | dialog, ID, width)

Sets the width of a `@RGNBUTTON` border. See [SETBUTTONBORDER](#) for a detailed discussion of options.

[SETHTCOLOR](#) (window | dialog, ID, color)

Sets the hot tracking color (color when the mouse is over the button) Is not applicable to buttons that use bitmaps.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the text of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a window, dialog or control. The dimensions returned is the bounding rectangle of the region control include the borders and caption.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a window, dialog or control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a window, dialog or control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the border, caption and non-client areas of a window, dialog or control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control. If the region button is using bitmaps the background color is ignored.

[SETCONTROLTEXT](#) window | dialog, ID, text

SETCONTROLTEXT is used to change the text of a control. If the control is a bitmap button or bitmap static control then the text is the path to a bitmap file.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus. The scrollbar control shows a highlight rectangle in the thumb slider.

[SETFONT](#) window | dialog, typeface, height, weight, flags, ID

Changes the font of all text in the control.

[SETSIZE](#) window | dialog, L, T, W, H, ID

Changes the size of the control. Do not use with a rgnbutton that has a region assigned. Doing so will distort the alignment of the text..

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use @SWHIDE to hide the control and @SWRESTORE to show the control

Notification messages

@BN_CLICKED

The control has been clicked.

@BN_DBLCLK

The control has been double clicked.

10.12.3.3 @SysButton Controls

About @SysButton controls

A SysButton is a control the user can click to provide input to an application. The SysButton control is Windows XP Theme compatible and its color can not be set manually.

Creating the control

Sysbutton controls are created either through the [Form Editor](#) or manually with the [CONTROL](#) statement. The following statement (with no flags applied)

```
CONTROL w1,@SYSBUTTON,"SysButton 1",25,15,100,25,0,w1_BUTTON1
```

results in this



@SysButton control styles

The following @SysButton style flags can be specified in the [CONTROL](#) statement or by ticking the corresponding check box in the control properties dialog of the [Form Editor](#):

@CTLBTNDEFAULT

Identifies this as the default push button in a dialog

@CTLBTNFLAT

Creates a flat button. This code

```
CONTROL w1,@SYSBUTTON,"Button 1 ",27,56,100,25,@CTLBTNFLAT,w1_BUTTON1
```

results in



@CTLBTNMULTI

Creates a multi line button control. Text is automatically word wrapped. This code

```
CONTROL w1,@SYSBUTTON,"SysButton with two lines ",27,56,100,40,@CTLBTNMULTI,w1_BUTTON1
```

results in



@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected. This code

```
CONTROL w1,@BUTTON,"Button 1 ",27,56,100,25,@DISABLE,w1_CLRBUTTON1
```

results in



. A control disabled in this manner can be enabled via the [ENABLECONTROL](#) command.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSTEMMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

@SysButton control functions and statements

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the text of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a window, dialog or control. The dimensions returned include the borders and caption.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a window, dialog or control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a window, dialog or control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the border, caption and non-client areas of a window, dialog or control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the border color of the control via the bg value provided there is a manifest file for the executable.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the text of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

SETFONT window | dialog, typeface, height, weight, flags, ID
Changes the font of all text in the control.

SETSIZE window | dialog, L, T, W, H, ID
Changes the size of the control. The scrollbar control is redrawn and the text will be formatted to match the new size of the control. The dimensions include the borders of the control.

SHOWWINDOW window | dialog, flags, ID
Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

Notification messages

@BN_CLICKED
The control has been clicked.

@BN_DBLCLK
The control has been double clicked.

10.12.4 Calendar Controls

About Calendar controls

A month calendar control implements a calendar-like user interface. The control below has been sized to contain two calendars:

November, 2014							December, 2014						
Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat
26	27	28	29	30	31	1		1	2	3	4	5	6
2	3	4	5	6	7	8	7	8	9	10	11	12	13
9	10	11	12	13	14	15	14	15	16	17	18	19	20
16	17	18	19	20	21	22	21	22	23	24	25	26	27
23	24	25	26	27	28	29	28	29	30	31	1	2	3
30							4	5	6	7	8	9	10
Today: 11/5/2014													

The above shows the following features:

- The current date is shown on a separate line at the bottom of the control. This is the default style.
- The "today circle" (actually a rectangle) appears around the current day, and beside the "Today" line as a visual cue. This is the default style.

Note

Windows does not support dates prior to 1601. The control is based on the Gregorian calendar and will not calculate dates that are consistent with the Julian calendar that was in use prior to 1753.

Selecting a day

By default, when a user clicks the arrow buttons at the top left or top right of the control, the control updates its display to show the previous or next month. The user can also perform the same action by clicking the partial months displayed before the first month and after the last month.

The following keyboard commands can also be used to move the selection. The calendar always scrolls as necessary to display the selected day.

Key	Function
Left arrow	Select the previous day.
Right arrow	Select the next day.
Up arrow	Select the same day in the previous week.
Down arrow	Select the same day in the next week.
PAGE UP	Select the same day in the previous month. (If that month does not have the day, the closest day is selected; for example, the selection moves from March 31 to February 28 or 29.)
PAGE DOWN	Select the same day in the next month.
HOME	Select the first day of the current month.
END	Select the last day of the current month.
CTRL + HOME	Scroll one month backward and select a day in the leftmost column.
CTRL + END	Scroll one month forward and select a day in the rightmost column.
CTRL + PAGE UP	Select the same day in an earlier month. The number of months by which the selection moves is the number of months displayed in the control. For example, if two months are displayed, the selection would move from June 6 to May 6.
CTRL + PAGE DOWN	Select the same day in an earlier month. The number of months by which the selection moves is the number of months displayed in the control. For example, if two months are displayed, the selection would move from June 6 to August 6.

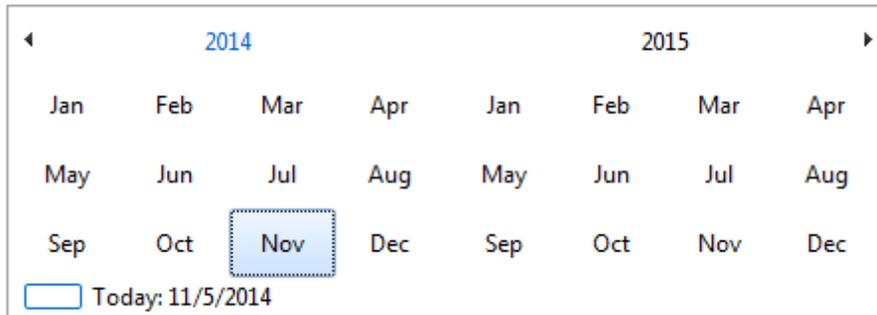
If a control is not using the **MCS_NOTODAY** style, the user can return to the current day by clicking the "Today" text at the bottom of the control. If the current day is not visible, the control updates its display to show it.

An application can change the number of months by which the control updates its display by using

the MCM_SETMONTHDELTA message. However, the PAGE UP and PAGE DOWN keys change the selected month by one, regardless of the number of months displayed or the value set by MCM_SETMONTHDELTA.

Selecting a nonadjacent month

When a user clicks the name of a displayed month, all months in the year are listed. The user can select a month on the list. If the user's selection is not visible, the control scrolls its display to show the chosen month. In the following screen shot, the control shows the months of two adjacent years.

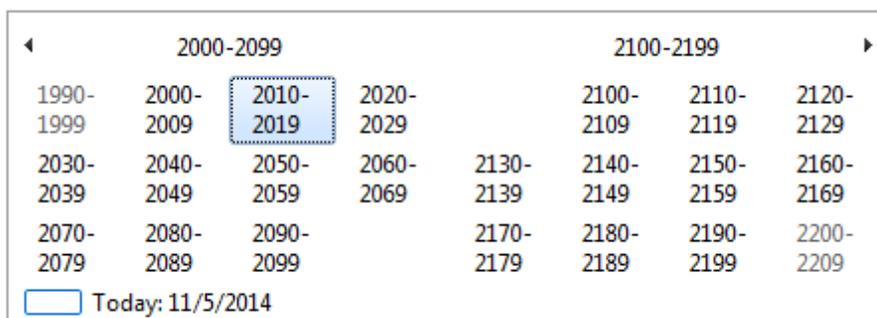


Selecting a different year

If the user clicks the year, a group of years is listed, and the user can select a different one, as shown in the following screen shot.



and clicking on the range of years above results in the following

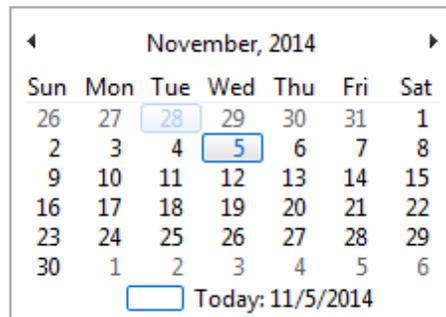


Creating the control

UINT = CalendarControl(win as WINDOW,l as INT,t as INT,w as INT,h as INT,flags as INT,exStyle as INT,id as UINT)

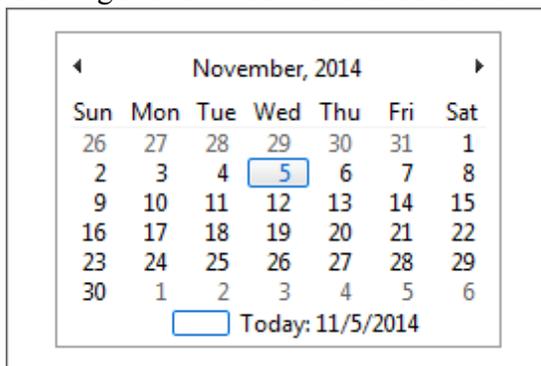
Calendar controls are created either through the [Form Editor](#) or manually with the [CalendarControl](#) statement. The following statement (with no flags applied)

```
CalendarControl w1,70,75,275,185,0,0,w1_CALENDAR1
```



results in this

Design Notes: The actual calendar(s) are a fixed size and use a non changeable font. When the user creates a Calendar control a container is created based upon the *width* and *height* used in the [CalendarControl](#) command. The control will then place as many calendars as possible in the container and centered. This makes it easy to place up to 12 months in one container. With the proper selection of *width* and *height* the months can be arranged in a 6x2, 2x6, 3x4, or 4x3 grid. To prevent the overlapping of the calendar control with other controls it is best to use the `@BORDER` style during development so that the actual container outline can be seen. The following shows the control created above with the `@BORDER` style:



This allows the User to see the buffer around the actual Calendar(s) and make adjustments as desired.

Calendar control styles

The following Calendar style flags can be specified in the [CalendarControl](#) statement or by ticking

the corresponding check box in the control properties dialog of the [Form Editor](#):

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

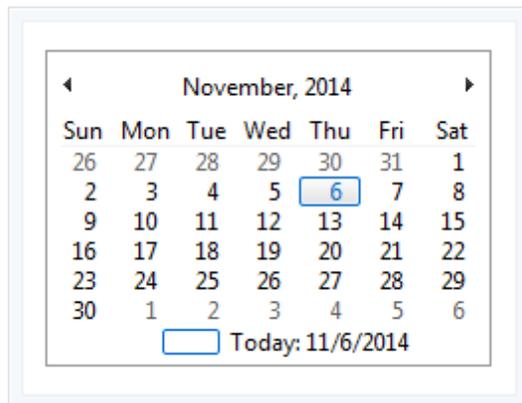
Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@TABSTOP

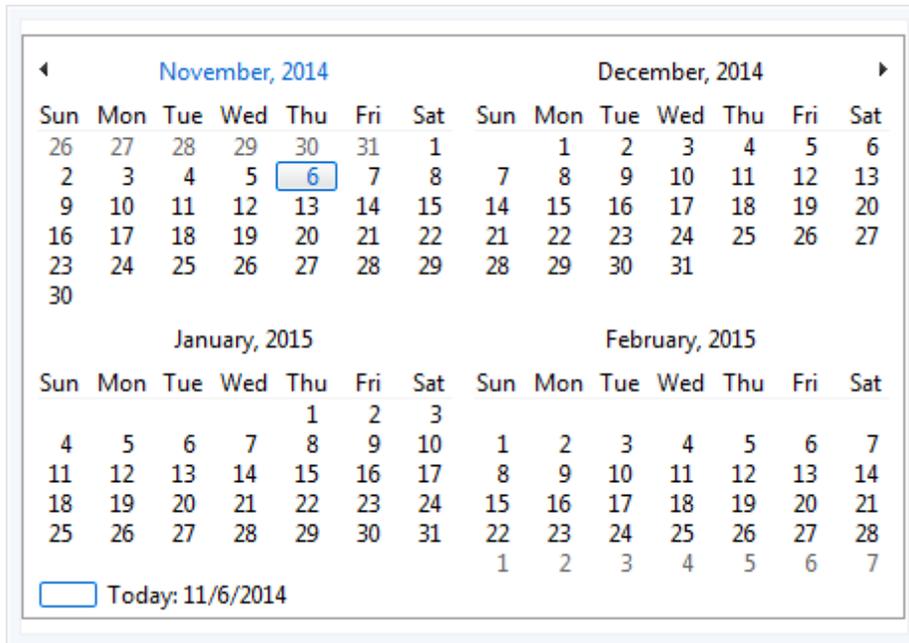
Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SIZE

Allows the edges of the control to be dragged to increase/decrease the size of the control. Using this style with the above example results in the following:



and if the control size is increased in both width and height by dragging the edges we can get this:

**@BORDER**

Outlines the container holding the calendar(s). Demonstrated previously.

@MCS_DAYSTATE

The control sends MCN_GETDAYSTATE notifications to request information about which days should be displayed in bold.

@MCS_MULTISELECT

The month calendar enables the user to select a range of dates within the control. By default, the maximum range is one week. You can change the maximum range that can be selected by using the MCM_SETMAXSELCOUNT message.

@MCS_NOTODAYCIRCLE

The control does not circle the "today" date.

@MCS_NOTODAY

The control does not display the "today" date at the bottom of the control.

@MCS_WEEKNUMBERS

The control displays week numbers (1-52) to the left of each row of days. Week 1 is defined as the first week that contains at least four days.

Calendar control functions and statements

UINT = [ccGetColor](#)(window | dialog, ID, index)

Returns the color of an element(index) of the calendar control.

MCSC_BACKGROUND - The background color (between months)

MCSC_TEXT - The dates

MCSC_TITLEBK - Background of the title

MCSC_TITLETEXT - Text color of the title.

MCSC_MONTHBK - Background within the calendar.

MCSC_TRAILINGTEXT - The text color of header & trailing days.

[ccGetCurSel](#)(window | dialog, ID, month, day, year)

Retrieves the currently selected date in a calendar control

INT = [ccGetFirstDayOfWeek](#)(window | dialog, ID)

Retrieves the first day of the week for a month calendar control.

[ccGetMinimumRect](#)(window | dialog, ID, rcRect)

Retrieves the minimum size required to display a full month in a month calendar control.

INT = [ccGetScrollDelta](#)(window | dialog, ID)

Retrieves the scroll rate for a month calendar control. The scroll rate is the number of months that the control moves its display when the user clicks a scroll button.

[ccGetToday](#)(window | dialog, ID, month, day, year)

Retrieves the date information for the date specified as "today" for a month calendar control.

[ccSetColor](#)(window | dialog, ID, index, clr)

Sets the color of an element of the calendar control. Where index is

MCSC_BACKGROUND - The background color (between months)



The following are disabled when the application has a MANIFEST file. In each example the designated color is set to red.

MCSC_TEXT - The dates

November, 2014							
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
44	26	27	28	29	30	31	1
45	2	3	4	5	6	7	8
46	9	10	11	12	13	14	15
47	16	17	18	19	20	21	22
48	23	24	25	26	27	28	29
49	30	1	2	3	4	5	6

MCSC_TITLEBK - Background of the title

November, 2014							
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
44	26	27	28	29	30	31	1
45	2	3	4	5	6	7	8
46	9	10	11	12	13	14	15
47	16	17	18	19	20	21	22
48	23	24	25	26	27	28	29
49	30	1	2	3	4	5	6

MCSC_TITLETEXT - Text color of the title.

November, 2014							
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
44	26	27	28	29	30	31	1
45	2	3	4	5	6	7	8
46	9	10	11	12	13	14	15
47	16	17	18	19	20	21	22
48	23	24	25	26	27	28	29
49	30	1	2	3	4	5	6

MCSC_MONTHBK - Background within the calendar.

November, 2014							
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
44	26	27	28	29	30	31	1
45	2	3	4	5	6	7	8
46	9	10	11	12	13	14	15
47	16	17	18	19	20	21	22
48	23	24	25	26	27	28	29
49	30	1	2	3	4	5	6

MCSC_TRAILINGTEXT - The text color of header & trailing days.



[ccSetCurSel](#)(window | dialog, ID, month, day, year)

Sets the currently selected date for a calendar control. If the specified date is not in view, the control updates the display to bring it into view.

[ccSetFirstDayOfWeek](#)(window | dialog, ID, day)

Sets the first day of the week for a calendar control. 0-7 (Monday-Sunday). Default is 7 (Sunday)

[ccSetScrollDelta](#)(window | dialog, ID, delta)

Sets the scroll rate for a calendar control. The scroll rate is the number of months that the control moves its display when the user clicks a scroll button. Default is 1.

[ccSetToday](#)(window|dialog, ID, month, day, year)

Sets the "today" selection for a calendar control.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam, ID

Sends a message to the control for advanced functionality.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETSIZE](#) window | dialog, L, T, W, H, ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

Notification messages

@MCN_SELCHANGE

Sent by a calendar control when the currently selected date or range of dates changes. This notification code is sent in the form of a `@IDNOTIFYCODE` message.

@MCN_SELECT

Sent by a calendar control when the user makes an explicit date selection within a month calendar control. This notification code is sent in the form of a `@IDNOTIFYCODE` message.

@MCN_GETDAYSTATE

Sent by a calendar control to request information about how individual days should be displayed. This notification code is sent only by calendar controls that use the `MCS_DAYSTATE` style, and it is sent in the form of a `@IDNOTIFYCODE` message. See the *calendar_demo.iwb* file for an example

10.12.5 Checkbox Controls

About Checkbox controls

A *checkbox* consists of a square box and an application-defined label, icon, or bitmap that indicates a choice the user can make by selecting the box. Applications typically display checkboxes to enable the user to choose one or more options that are not mutually exclusive.

checkbox1

Creating the control

Checkbox controls are created either through the [Form Editor](#) or manually with the [CONTROL](#) statement.

```
CONTROL win,@CHECKBOX,"Checkbox 1",73,84,60,25,0,win_CHECKBOX1
```

Checkbox control styles

The following Checkbox style flags can be specified in the [CONTROL](#) statement or by ticking the corresponding check box in the control properties dialog of the [Form Editor](#):

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the [@GROUP](#) style after the first control belong to the same group. The next control with the [@GROUP](#) style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

@CTLBTNMULTI

Creates a multi line button control.

```
 checkbox
1
```

@LEFTTEXT|

Places the text area to the left of the box.

```
checkbox1 
```

BS_LEFT

Aligns the text to the left of the text area.

```
 checkbox
1
```

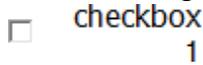
BS_CENTER

Aligns the text to the center of the text area.

```
 checkbox
1
```

BS_RIGHT

Aligns the text to the right of the text area.

**Checkbox control functions and statements**

[SETSTATE](#) window | dialog, ID, state

Sets or resets a checkbox control. State can either be 0 to uncheck the control or 1 to check the control.

state = [GETSTATE](#) (window | dialog, ID)

Returns the state of a checkbox control. Returns 1 if control is checked and 0 if control is unchecked.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wParam, lParam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control. If there is a manifest file the text color does not change.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight, flags, ID

Changes the font of all text in the control.

[SETSIZE](#) window | dialog, L, T, W, H, ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

Notification messages

@BN_CLICKED

The control has been clicked. Sent to the parent.

10.12.6 Combobox Controls

About combo box controls

A *combo box* is a unique type of control that combines much of the functionality of a list box and an edit control.

A combo box consists of a list and a selection field. The list presents the options a user can select and the selection field displays the current selection. Except in drop-down list boxes, the selection field is an edit control and can be used to enter text not in the list.

Creating the control

Combo box controls are created either through the [Form Editor](#) or manually with the [CONTROL](#) statement.

Combo box control styles

The following combo box style flags can be specified in the [CONTROL](#) statement or by ticking the corresponding check box in the control properties of the [Form Editor](#):

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the [@GROUP](#) style after the first control belong to the same group. The next control with the [@GROUP](#) style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSTEMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

@CTCOMBODROPDOWN

Similar to [@CTCOMBOSIMPLE](#), except that the list box is not displayed unless the user selects an icon next to the edit control.

@CTCOMBODROPLIST

Similar to [@CTCOMBODROPDOWN](#), except that the edit control is replaced by a static text item that displays the current selection in the list box.

@CTCOMBOSIMPLE

Displays the list box at all times. The current selection in the list box is displayed in the edit control.

@CTCOMBOSORT

Automatically sorts strings added to the list box.

@CTCOMBOAUTOHSCROLL

Automatically scrolls the text in an edit control to the right when the user types a character at the end of the line. If this style is not set, only text that fits within the rectangular boundary is allowed.

In addition to the combo box styles, the following window styles can also be specified:

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@HSCROLL

The control has a horizontal scroll bar

@VSCROLL

The control has a vertical scrollbar

Combo box control functions and statements

The following IW BASIC functions and statements are used to communicate with the combo box control.

[SETSELECTED](#) window | dialog, ID, position

Sets the currently selected item in a list or combo box control. The position value is zero-based and the list or combo box must contain a string at that position.

[GETSELECTED](#) (window | dialog, ID)

Returns the zero-based index of the currently selected item in the list box of a combo or single selection list box. Returns -1 if no item is selected. ID must be a list box or combo box.

[ISSELECTED](#) (window | dialog, ID, position)

Returns TRUE if the string at position is selected in a multi-selection list or combo box.

[GETSTRINGCOUNT](#) (window | dialog, ID)

Returns the number of strings in a list box or combo box control.

[GETSTRING](#) (window | dialog, ID, position)

Returns the string at position in a list box or combo box. Position is a zero based index.

[ADDSTRING](#) window | dialog, ID, string

Adds a string to a list box or combo box control. String is added to the end of the list unless sorting is specified in the style of the control.

[INSERTSTRING](#) window | dialog, ID, position, text

Inserts a string into a combo or list box control at position. All other strings are moved down by one position.

[DELETSTRING](#) window | dialog, ID, position

Removes a string from a list box or combo box control. Remaining strings are moved up to fill the empty position.

[SETHORIZEXTENT](#) window | dialog, ID, width

Sets the horizontal scroll width of a list box or list box portion of a combo box. Control must have been created with the @HSCROLL style. The width is specified in pixels.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

GETCONTROLTEXT is used to retrieve the text in the edit window portion of the control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text

SETCONTROLTEXT is used to change the typed text in the edit window portion of the control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

SETFONT window | dialog, typeface, height, weight, flags, ID
Changes the font of all text in the control.

SETSIZE window | dialog, L, T, W, H, ID
Changes the size of the control.

SHOWWINDOW window | dialog, flags, ID
Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

Notification messages

An combo box control sends notification messages to the parent window or dialog in the `@NOTIFYCODE` variable. The ID of the control is found in `@CONTROLID`.

@CBNDBLCLICK

Indicates the user has double-clicked a list item in a simple combo box.

@CBNERRSPACE

Indicates the combo box cannot allocate enough memory to carry out a request, such as adding a list item.

@CBNKILLFOCUS

Indicates the combo box is about to lose the input focus.

@CBNSETFOCUS

Indicates the combo box has received the input focus.

@CBNDROPDOWN

Indicates the list in a drop-down combo box or drop-down list box is about to open.

@CBNCLOSEUP

Indicates the list in a drop-down combo box or drop-down list box is about to close.

@CBNEDITCHANGE

Indicates the user has changed the text in the edit control of a simple or drop-down combo box. This notification message is sent after the altered text is displayed.

@CBNEDITUPATE

Indicates the user has changed the text in the edit control of a simple or drop-down combo box. This notification message is sent before the altered text is displayed.

@CBNSELCHANGE

Indicates the current selection has changed.

@CBNSELENDOK

Indicates that the selection made drop down list, while it was dropped down, should be accepted.

@CBNSELEND CANCEL

Indicates that the selection made in the drop down list, while it was dropped down, should be ignored.

10.12.7 ComboboxEx Controls - WIP

About comboboxex controls

ComboBoxEx controls are combo box controls that provide native support for item images.

Creating the control

UINT = ComboBoxEx(win as WINDOW,l as INT,t as INT,w as INT,h as INT,flags as INT,exStyle as INT,id as UINT)

Combo boxex control styles

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

@CTCOMBODROPDOWN

Similar to @CTCOMBOSIMPLE, except that the list box is not displayed unless the user selects an icon next to the edit control.

@CTCOMBODROPLIST

Similar to @CTCOMBODROPDOWN, except that the edit control is replaced by a static text item that displays the current selection in the list box.

@CTCOMBOSIMPLE

Displays the list box at all times. The current selection in the list box is displayed in the edit control.

@CTCOMBOSORT

Automatically sorts strings added to the list box.

@CTCOMBOAUTOHSCROLL

Automatically scrolls the text in an edit control to the right when the user types a character at the end of the line. If this style is not set, only text that fits within the rectangular boundary is allowed.

In addition to the combo box styles, the following window styles can also be specified:

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@HSCROLL

The control has a horizontal scroll bar

@VSCROLL

The control has a vertical scrollbar

Combo box control functions and statements

[cbeAddString](#)(window | dialog, ID, text, OPT image=-2, OPT selimage=-2)

Adds a string item to a ComboBoxEx control. String is added to the end of the list.

[cbeDeleteString](#)(window | dialog, ID, pos)

Removes a string from a ComboBoxEx control. Remaining strings are moved up to fill the empty position

INT = [cbeGetSelected](#)(window | dialog, ID)

Returns the zero-based index of the currently selected item in a ComboBoxEx control.

STRING = [cbeGetString](#)(window | dialog, ID, pos)

Returns a string in a ComboBoxEx control.

INT = [cbeGetStringCount](#)(window | dialog, id)

Returns the number of strings in a ComboBoxEx control.

[cbeInsertString](#)(window | dialog, ID, text, pos, OPT image=-2, OPT selimage=-2)

Inserts a string into a ComboBoxEx control.

[cbeSetImageList](#)(window | dialog, ID, hIml)

Sets an image list for a ComboBoxEx control.

[cbeSetIndent](#)(window | dialog, ID, pos, indent)

Sets the number of indent spaces to display for the item in the ComboBoxEx control. Each indentation equals 10 pixels.

[cbeSetSelected](#)(window | dialog, ID, pos)

Sets the currently selected item in a ComboBoxEx control.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the text in the edit window portion of the control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wParam, lParam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text
Used to change the text in the edit window portion of the control.

[SETFOCUS](#) window | dialog, ID
Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight, flags, ID
Changes the font of all text in the control.

[SETSIZE](#) window | dialog, L, T, W, H, ID
Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID
Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

Notification messages

10.12.8 Date/TimePicker Controls - WIP

About Date/TimePicker controls

A *date and time picker (DTP)* control provides a simple and intuitive interface through which to exchange date and time information with a user.

Creating the control

UINT = DateTimePicker(win as WINDOW, title as STRING, l as INT, t as INT, w as INT, h as INT, flags as INT, exStyle as INT, id as UINT)

Date/TimePicker control styles

@DTS_UPDOWN

Use UPDOWN instead of MONTHCAL

@DTS_SHOWNONE

Allow a NONE selection

@DTS_SHORTDATEFORMAT

Use the short date format

@DTS_LONGDATEFORMAT

Use the long date format

@DTS_TIMEFORMAT

Use the time format

@DTS_APPCANPARSE

Allow user entered strings (app MUST respond to DTN_USERSTRING)

@DTS_RIGHTALIGN

right-align popup instead of left-align it

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

```
if getstate( t,1039 ) then style+="@DTS_UPDOWN|"
if getstate( t,1040 ) then style+="@DTS_SHOWNONE|"
if getstate( t,1041 ) then style+="@DTS_APPCANPARSE|"
if getstate( t,1042 ) then style+="@DTS_RIGHTALIGN|"
if getstate( t,1043 ) then style+="@DTS_LONGDATEFORMAT|"
if getstate( t,1044 ) then style+="@DTS_TIMEFORMAT|"
```

Date/Time Picker control functions and statements

UINT = [dtpGetMCColor](#)(window | dialog, ID, index)

Retrieves the color for a given portion of the month calendar within a date and time picker (DTP) control.

SYSTEMTIME = [dtpGetSystemTime](#)(window | dialog, ID)

Retrieves the currently selected time from a date and time picker (DTP) control and places it in a SYSTEMTIME UDT.

[dtpSetFormat](#)(window | dialog, ID, format)

Sets the display of a date and time picker (DTP) control based on the given format string.

[dtpSetMCColor](#)(window | dialog, ID, index, clr)

Sets the color for a given portion of the month calendar within a date and time picker (DTP) control.

[dtpSetSystemTime](#)(window | dialog, ID, tm)

Sets a date and time picker (DTP) control to a given date and time.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight, flags, ID

The text printed in a control will use the default font specified in the display control panel unless changed with the [SETFONT](#) statement.

[SETSIZE](#) window | dialog, L, T, W, H, ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

Notification messages

DTN_DATETIMECHANGE

10.12.9 Edit Controls - WIP

About edit controls

An *edit control* is a rectangular control window typically used in a dialog box to permit the user to enter and edit text from the keyboard. Edit controls are single font, single text color controls. If you need more advanced word processing features see the documentation on *rich edit* controls.

Creating the control

Edit controls are created either through the [Form Editor](#) or manually with the [CONTROL](#) statement.

Edit control styles

The following edit style flags can be specified in the [CONTROL](#) statement or by ticking the corresponding check box in the control properties of the [Form Editor](#):

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSTEMMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

@CTEDITLEFT

Text is left justified in the edit control

@CTEDITRIGHT

Text is right justified in the edit control

@CTEDITMULTI

Designates a multiline edit control. The default is single-line edit control. When the multiline edit control is in a dialog box, the default response to pressing the ENTER key is to activate the default button. To use the ENTER key as a carriage return, use the @CTEDITRETURN style. When the multiline edit control is not in a dialog box and the @CTEDITAUTOV style is specified, the edit control shows as many lines as possible and scrolls vertically when the user presses the ENTER key. If you do not specify @CTEDITAUTOV, the edit control shows as many lines as possible and beeps if the user presses the ENTER key when no more lines can be displayed. If you specify the @CTEDITAUTOH style, the multiline edit control automatically scrolls horizontally when the caret goes past the right edge of the control. To start a new line, the user must press the ENTER key. If you do not specify @CTEDITAUTOH, the control automatically wraps words to the beginning of the next line when necessary. A new line is also started if the user presses the ENTER key. The window size determines the position of the word wrap. If the window size changes, the word wrapping position changes and the text is redisplayed. Multiline edit controls can have scroll bars. An edit control with scroll bars processes its own scroll bar messages. Note that edit controls without scroll bars scroll as described in the previous paragraphs and process any scroll messages sent by the parent window.

@CTEDITPASS

Displays an asterisk (*) for each character typed into the edit control.

@CTEDITCENTER

Text is centered within the edit control

@CTEDITRO

The edit control is read only and text can be displayed but not entered

@CTEDITAUTOH

Automatically scrolls text to the right by 10 characters when the user types a character at the end of the line. When the user presses the ENTER key, the control scrolls all text back to position zero.

@CTEDITAUTOV

Automatically scrolls text up one page when the user presses the ENTER key on the last line.

@CTEDITRETURN

Specifies that a carriage return be inserted when the user presses the ENTER key while entering text into a multiline edit control in a dialog box. If you do not specify this style, pressing the ENTER key has the same effect as pressing the dialog box's default push button. This style has no effect on a single-line edit control.

@CTEDITNUMBER

Restricts text entered in a single line edit control to numerals only (0 - 9)

In addition to the edit styles the following window styles can also be specified:

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@HSCROLL

The control has a horizontal scroll bar

@VSCROLL

The control has a vertical scrollbar

Edit control functions and statements

The following IWBASIC functions and statements are used to communicate with the edit control.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

GETCONTROLTEXT is used to retrieve the typed text of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text

SETCONTROLTEXT is used to change the typed text of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight , flags , ID

Changes the font of all text in the control.

[SETSIZE](#) window | dialog, L, T, W, H , ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use @SWHIDE to hide the control and @SWRESTORE to show the control

Clipboard operations

[CONTROLCMD](#) window | dialog, ID, @EDCUT

Use this command to delete (cut) the current selection (if any) in the edit control and copy the deleted text to the Clipboard.

[CONTROLCMD](#) window | dialog, ID, [@EDCOPY](#)

Use this command to copy the current selection (if any) in the edit control to the Clipboard.

[CONTROLCMD](#) window | dialog, ID, [@EDPASTE](#)

Use this command to insert the data from the Clipboard into the edit control at the insertion point, the location of the caret. Data is inserted only if the Clipboard contains data in text format.

Line operations

count = [CONTROLCMD](#) (window | dialog, ID, [@EDGETLINECOUNT](#))

Use this function to retrieve the number of lines in the edit control

[CONTROLCMD](#) (window | dialog, ID, [@EDGETLINE](#), linenum, line\$ {,cchLine})

Use this function to retrieve a line of text from the edit control.

Linenum is the 0 based index of the line to retrieve, *cchLine* is the size of the *line\$* STRING variable, defaults at 255.

line = [CONTROLCMD](#) (window | dialog, ID, [@EDGETFIRSTLINE](#))

Use this function to retrieve the zero-based index of the uppermost visible line.

line = [CONTROLCMD](#) (window | dialog, ID, [@EDLINEFROMCHAR](#), index)

Use this function to retrieve the line number of the line that contains the specified character index. Index is the number of characters from the beginning of the edit control.

char_index = [CONTROLCMD](#) (window | dialog, ID, [@EDCHARFROMLINE](#), linenum)

Use this function to retrieve the character index of the first character of the specified line.

length = [CONTROLCMD](#) (window | dialog, ID, [@EDGETLINELENGTH](#), index)

Use this function to retrieve the length of a line in a edit control. When [@EDGETLINELENGTH](#) is called for a multiple-line edit control, the return value is the length (in bytes) of the line specified by index. When [@EDGETLINELENGTH](#) is called for a single-line edit control, the return value is the length (in bytes) of the text in the edit control.

Index specifies the character index of a character in the line whose length is to be retrieved. If this parameter is -1, the length of the current line (the line that contains the caret) is returned

Selection operations

[CONTROLCMD](#) window | dialog, ID, [@EDGETSELECTION](#), varStart, varEnd

Use this command to retrieve the current selection of the edit control.

varStart and varEnd must be of type INT. The zero-based index of the first and last characters selected are copied into the two variables. The selection includes everything if varStart =0 and varEnd = -1.

[CONTROLCMD](#) window | dialog, ID, [@EDDELETESEL](#)

Use this statement to delete the current selection. The deletion performed by [@EDDELETESEL](#) can be undone by using [@EDUNDO](#)

[CONTROLCMD](#) window | dialog, ID, [@EDSETSELECTION](#), start, end

Use this statement to set the current selection in the edit control
start and end are the zero-based character indexes of the selection. If start = 0 and end = -1 then all of the text is selected.

[CONTROLCMD](#) window | dialog, ID, @EDREPLACESEL, text\$

Use this statement to replace the current selection text

Editing operations

[CONTROLCMD](#) window | dialog, ID, @EDUNDO

Use this statement to undo the last editing operation. An undo operation can also be undone. For example, you can restore deleted text with the first call to Undo. As long as there is no intervening edit operation, you can remove the text again with a second call to Undo.

return = [CONTROLCMD](#) (window | dialog, ID, @EDCANUNDO)

Use this function to determine if the last editing operation can be undone. Returns 0 if the last operation cannot be undone.

[CONTROLCMD](#) window | dialog, ID, @EEMPTYUNDO

Use this statement to reset (clear) the undo flag of this edit control. The control will now be unable to undo the last editing operation. The undo flag is set whenever an operation within the rich edit control can be undone.

General operations

return = [CONTROLCMD](#) (window | dialog, ID, @EDGETMODIFIED)

Use this function to determine if the contents of the edit control have changed. Returns 1 if the contents have been modified, 0 otherwise.

[CONTROLCMD](#) window | dialog, ID, @EDSETMODIFIED, mod

Sets the modified flag of the edit control.

Mod can be 0 to reset the flag or 1 to set it.

length = [CONTROLCMD](#) (window | dialog, ID, @EDGETLIMITTEXT)

Use this function to get the text limit for this edit control. The text limit is the maximum amount of text, in bytes, the edit control can accept either through pasting or typing.

[CONTROLCMD](#) window | dialog, ID, @EDSETLIMITTEXT, length

Use this function to set the text limit for this edit control. The text limit is the maximum amount of text, in bytes, the edit control can accept either through pasting or typing. The default limit is 32767 bytes for multi line controls.

[CONTROLCMD](#) window | dialog, ID, @EDSETMARGINS, left, right

Use this statement to set the visible left and right margins of the edit control.

Left and right are specified in pixels.

Notification messages

An edit control sends notification messages to the parent window or dialog in the

@NOTIFYCODE variable. The ID of the control is found in @CONTROLID. The following

notification messages are supported:

@ENKILLFOCUS

Control has lost input focus

@ENSETFOCUS

Control has been given the input focus

@ENERRSPACE

Control could not complete an operation because there was not enough memory available

@ENMAXTEXT

While inserting text, the user has exceeded the specified number of characters for the edit control. Insertion has been truncated. This message is also sent either when an edit control does not have the @CTEDITAUTOH style and the number of characters to be inserted exceeds the width of the edit control or when an edit control does not have the @CTEDITAUTOV style and the total number of lines to be inserted exceeds the height of the edit control.

@ENUPDATE

The contents of the control are about to change

@ENCHANGE

The contents of the control have changed.

@ENHSCROLL

The user has clicked the edit control's horizontal scroll bar. Windows sends this message before updating the control.

@ENVSCROLL

The user has clicked the edit control's vertical scroll bar. Windows sends this message before updating the control.

@ENTABKEY

The user has pressed the TAB key while the control has input focus. Only sent if enabled by the [SETCONTROLNOTIFY](#) command.

@ENENTERKEY

The user has pressed the ENTER key while the control has input focus. Only sent if enabled by the [SETCONTROLNOTIFY](#) command.

10.12.1 Group Controls - WIP

About Group controls

Creating the control

Group controls are created either through the [Form Editor](#) or manually with the [CONTROL](#) statement.

Group control styles

The following Group style flags can be specified in the [CONTROL](#) statement or by ticking the corresponding check box in the control properties dialog of the [Form Editor](#):

[@DISABLE](#)

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

[@SYSTEMMENU](#)

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

Group control functions and statements

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wParam, lParam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the title of the control. If there is a manifest file the text color does not change.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight, flags, ID

Changes the font of the caption in the control.

[SETSIZE](#) window | dialog, L, T, W, H, ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use @SWHIDE to hide the control and @SWRESTORE to show the control

Notification messages

10.12.1 Header Controls - WIP

About Header controls

A header control is a window that is usually positioned above columns of text or numbers. It contains a title for each column, and it can be divided into parts.

Creating the control

UINT = HeaderControl(win as WINDOW, l as INT, t as INT, w as INT, h as INT, flags as INT, exStyle as INT, id as UINT)

Header control styles

@HDS_HORZ

Creates a header control with a horizontal orientation.

@HDS_BUTTONS

Each item in the control looks and behaves like a push button.

@HDS_HOTTRACK

Enables hot tracking.

@HDS_HIDDEN

Indicates a header control that is intended to be hidden.

@HDS_DRAGDROP

Allows drag-and-drop reordering of header items.

@HDS_FULLDRAG

Causes the header control to display column contents even while the user resizes a column.

Header control functions and statements

[hcDeleteItem](#)(window | dialog, ID, index as INT)

Deletes an item from a header control.

INT = [hcGetItemCount](#)(window | dialog, ID)

Retrieves a count of the items in a header control

UINT = [hcGetItemData](#)(window | dialog, ID, index)

Returns the application-defined item data associated with an item in the header control.

WINRECT = [hcGetItemRect](#)(window | dialog, ID, index)

Retrieves the bounding rectangle for a specified item in a header control.

STRING = [hcGetItemText](#)(window | dialog, ID, index)

Returns the text of an item in the header control.

INT = [hcGetItemWidth](#)(window | dialog, ID, index)

Returns the width, in pixels of an item in the header control.

[hcInsertItem](#)(window | dialog, ID, index, text, width, OPT image=-2)

Inserts an item into the header control.

[hcSetImageList](#)(window | dialog, ID, hIml)

Sets the image list used by the header control.

[hcSetItemData](#)(window | dialog, ID, index, value)

Sets the application-defined item data associated with an item in the header control.

[hcSetItemJustify](#)(window | dialog, ID, index, justify)

Sets the justification of the text of an item in the header control.

[hcSetItemText](#)(window | dialog, ID, index, text)

Changes the text of an item in the header control.

[hcSetItemWidth](#)(window | dialog, ID, index, width)

Sets the width, in pixels, of an item in the header control.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wParam, lParam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight, flags, ID

The text printed in a control will use the default font specified in the display control panel unless changed with the [SETFONT](#) statement.

[SETSIZE](#) window | dialog, L, T, W, H, ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use [@SWHIDE](#) to hide the control and [@SWRESTORE](#) to show the control

Notification messages

HDN_BEGINDRAG

HDN_BEGINFILTEREDIT

HDN_BEGINTRACK

HDN_DIVIDERDBLCLICK

HDN_DROPDOWN

HDN_ENDDRAG

HDN_ENDFILTEREDIT

HDN_FILTERBTNCLICK

HDN_FILTERCHANGE

HDN_GETDISPINFO

HDN_ITEMCHANGED

HDN_ITEMCHANGING

HDN_ITEMCLICK

HDN_ITEMDBLCLICK

HDN_ITEMKEYDOWN

HDN_ITEMSTATEICONCLICK

HDN_OVERFLOWCLICK

HDN_TRACK NM_CUSTOMDRAW (header)

NM_RCLICK (header)

NM_RELEASEDCAPTURE (header)

10.12.11 IP Controls - WIP

About IP controls

An Internet Protocol (IP) address control allows the user to enter an IP address in an easily understood format.

Creating the control

UINT = IPControl(win as WINDOW, l as INT, t as INT, w as INT, h as INT, flags as INT, exStyle as INT, id as UINT)

IP control styles

IP control functions and statements

[IPClearAddress](#)(window | dialog, ID)

Clears the contents of the IP address control.

[IPGetAddress](#)(window | dialog, ID, f1, f2, f3, f4)

Returns the IP address contained in the control.

UINT = [IPGetAddressDword](#)(window | dialog, ID)

Gets the address values for all four fields in the IP address control.

INT = [IPIsBlank](#)(window | dialog, ID)

Determines if all fields in the IP address control are blank.

[IPSetAddress](#)(window | dialog, ID, f1, f2, f3, f4)

Sets the address values for all four fields in the IP address control.

[IPSetAddressDword](#)(window | dialog, ID, address)

Sets the address values for all four fields in the IP address control.

[IPSetRange](#)(window | dialog, ID, field, low, high)

Sets the valid range for the specified field in the IP address control.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight , flags , ID

The text printed in a control will use the default font specified in the display control panel unless changed with the [SETFONT](#) statement.

[SETSIZE](#) window | dialog, L, T, W, H , ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use @SWHITE to hide the control and @SWRESTORE to show the control

Notification messages

IPN_FIELDCHANGED

10.12.1: ListBox Controls - WIP

About list box controls

A *list box* is a control window that contains a list of items from which the user can choose.

List box items are represented by text strings. If the list box is not large enough to display all the list box items at once, the list box can provide a scroll bar. The user maneuvers through the list box items, scrolling the list when necessary, and selects or removes the selection from items. Selecting a list box item changes its visual appearance, usually by changing the text and background colors to the colors specified by the operating system metrics for selected items. When the user selects an item or removes the selection from an item, Windows sends a notification message to the parent window of the list box.

Creating the control

List box controls are created in the same manner as the standard control types, either through the [Form Editor](#) or manually with the [CONTROL](#) statement.

List box control styles

The following list box style flags can be specified in the [CONTROL](#) statement or by ticking the corresponding check box in the control properties of the [Form Editor](#):

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the [@GROUP](#) style after the first control belong to the same group. The next control with the [@GROUP](#) style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

```
if getstate( t,1039 ) then style+="@BORDER|"  
  
'style+="@CTEDITLEFT|"
```

@CTLISTEXTENDED

Allows multiple items to be selected by using the SHIFT key and the mouse or special key combinations.

@CTLISTMULTI

Turns string selection on or off each time the user clicks or double-clicks a string in the list box. The user can select any number of strings.

@CTLISTSORT

Sorts strings in the list box alphabetically.

@CTLISTSTANDARD

Sorts strings in the list box alphabetically. The parent window receives an input message whenever the user clicks or double-clicks a string. The list box has borders on all sides.

@CTLISTNOTIFY

Notifies the parent window with an input message whenever the user clicks or double-clicks a string in the list box.

@CTLISTTABS

Enables a list box to recognize and expand tab characters when drawing its strings.

@CTLISTCOLUMNS

Specifies a multicolumn list box that is scrolled horizontally. The [SETLBCOLWIDTH](#) statement sets the width of the columns.

In addition to the list box styles, the following window styles can also be specified:

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@HSCROLL

The control has a horizontal scroll bar

@VSCROLL

The control has a vertical scrollbar

List box control functions and statements

The following IW BASIC functions and statements are used to communicate with the list box control.

[SETSELECTED](#) window | dialog, ID, position

Sets the currently selected item in a list or combo box control. The position value is zero-based and the list or combo box must contain a string at that position.

[GETSELECTED](#) (window | dialog, ID)

Returns the zero-based index of the currently selected item in the list box of a combo or single selection list box. Returns -1 if no item is selected. ID must be a list box or combo box.

[ISSELECTED](#) (window | dialog, ID, position)

Returns TRUE if the string at position is selected in a multi-selection list or combo box.

[GETSTRINGCOUNT](#) (window | dialog, ID)

Returns the number of strings in a list box or combo box control.

[GETSTRING](#) (window | dialog, ID, position)

Returns the string at position in a list box or combo box. Position is a zero based index.

[ADDSTRING](#) window | dialog, ID, string

Adds a string to a list box or combo box control. String is added to the end of the list unless sorting is specified in the style of the control.

[INSERTSTRING](#) window | dialog, ID, position, text

Inserts a string into a combo or list box control at position. All other strings are moved down by one position.

[DELETESTRING](#) window | dialog, ID, position

Removes a string from a list box or combo box control. Remaining strings are moved up to fill the empty position.

[SETLBCOLWIDTH](#) window | dialog, ID, width

Sets the width of columns in a multi-column list box. The list box must have been created with the style `@CTLISTMULTI` either in the CONTROL statement or by selecting the multicolumn checkbox in the [Form Editor](#).

[SETHORIZEXTENT](#) window | dialog, ID, width

Sets the horizontal scroll width of a list box or list box portion of a combo box. Control must have been created with the `@HSCROLL` style. The width is specified in pixels.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)
Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)
Used to retrieve the hidden caption of the control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)
Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)
Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)
Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)
Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID
Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg
Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text
Used to change the hidden caption of the control.

[SETFOCUS](#) window | dialog, ID
Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight , flags , ID
Changes the font of all text in the control.

[SETSIZE](#) window | dialog, L, T, W, H , ID
Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID
Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

sendmessage dy,LB_RESETCONTENT,0,0,747

LB_GETTOPINDEX

LB_TETTOPINDEX

Notification messages

An list box control sends notification messages to the parent window or dialog in the @NOTIFYCODE variable. The ID of the control is found in @CONTROLID. To receive @LBNDLCLK or click messages it is necessary to create the control with the @CTLISTNOTIFY style. The following notification messages are defined:

@LBNDLCLK

The user double-clicks an item in the list box.

@LBNERRSPACE

The list box cannot allocate enough memory to fulfill a request.

@LBNKILLFOCUS

The list box loses the keyboard focus.

@LBNSSETFOCUS

The list box receives the keyboard focus.

@LBNSELCHANGE

The selection in a list box is about to change.

@LBNSELCANCEL

The user cancels the selection of an item in the list box.

10.12.14 ListView Controls - WIP

About list view controls

A list view control is a window that displays a collection of items, each item consisting of a label. List view controls provide several ways of arranging items and displaying individual items. For example, additional information about each item can be displayed in columns to the right of the label. List view controls can be shown in report mode in which case a selectable header is used. The Windows explorer in 'detail' view is one example of a list view control. Image lists and icon views are not currently supported directly in IWBASIC. This functionality will be added in a future version. You can create image lists with the Windows API and the DLL comctl32.dll.

Creating the control

List view controls are created in the same manner as the standard control types, either through the [Form Editor](#) or manually with the [CONTROL](#) statement.

List view control styles

The following list view style flags can be specified in the [CONTROL](#) statement or by ticking the corresponding check box in the control properties of the [Form Editor](#):

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

```
if getstate( t,1039 ) then style+="@BORDER!"
```

@LVSALIGNLEFT

Specifies that items are left-aligned in icon and small icon view.

@LVSALIGNTOP

Specifies that items are aligned with the top of the control in icon and small icon view.

@LVSAUTOARRANGE

Specifies that icons are automatically kept arranged in icon view and small icon view.

@LVSEDLABELS

Allows item text to be edited in place. The parent window must process the LVN_ENDLABELEDIT notification message.

@LVSICON

Specifies icon view.

@LVSLIST

Specifies list view.

@LVSNOCOLUMNHEADER

Specifies that a column header is not displayed in report view. By default, columns have headers in report view.

@LVSNO LABELWRAP

Displays item text on a single line in icon view. By default, item text can wrap in icon view.

@LVSNO SCROLL

Disables scrolling. All items must be within the client area.

@LVSNO SORTHEADER

Specifies that column headers do not work like buttons. This style is useful if clicking a column header in report view does not carry out an action, such as sorting.

@LVSREPORT

Specifies report view.

@LVSSHOWSELALWAYS

Always show the selection, if any, even if the control does not have the focus.

@LVSSINGLESEL

Allows only one item at a time to be selected. By default, multiple items can be selected.

@LVSSMALLICON

Specifies small icon view.

@LVSSORTASCENDING

Sorts items based on item text in ascending order.

@LVSSORTDESCENDING

Sorts items based on item text in descending order.

In addition to the list view styles, the following window styles can also be specified:

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the **@GROUP** style after the first control belong to the same group. The next control with the **@GROUP** style starts the next group (that is, one group ends where the next begins)

@HSCROLL

The control has a horizontal scroll bar

@VSCROLL

The control has a vertical scrollbar

Controlling the list view control.

The list view control is accessed through the [CONTROLCMD](#) statement/function.
[CONTROLCMD](#) can be used as a statement or function depending on the command issued.

Statements and Functions

[CONTROLCMD](#) window | dialog, ID, @LVDELETEALL

Use this statement to delete all items in the list view control

[CONTROLCMD](#) window | dialog, ID, @LVDELETECOLUMN, col

Use this statement to delete a column in the list view control.
 col is the zero-based index of the column to delete.

[CONTROLCMD](#) window | dialog, ID, @LVDELETEITEM, position

Use this statement to delete a item in the list view.
 position is the zero-based index of the item to delete.

[CONTROLCMD](#) window | dialog, ID, @LVINSERTITEM, position, item\$

Use this statement to insert an item into the list view control.
 Position is the zero-based index of the item to insert.
 Item\$ is the text of the item

[CONTROLCMD](#) window | dialog, ID, @LVINSERTCOLUMN, column, text\$

Use this statement to insert a column into list control. The control must have been created in report view.
 Column is the zero-based index of the new column
 Text\$ is the text of the column.

[CONTROLCMD](#) window | dialog, ID, @LVSETTEXT, item, subitem, text\$

Use this statement to set the text of an item or subitem
item is the zero-based index of the item
subitem is the ones-based index of the subitem or 0 to change the item text
text\$ is the new text.

[CONTROLCMD](#)(window | dialog, ID, @LVGETTEXT, item, subitem, text\$, {cchText})

Use this function to retrieve the text of an item or subitem.
item is the zero-based index of the item
 subitem is the ones-based index of the subitem or 0 to retrieve the item text
text\$ is a variable of type STRING to store retrieved text.
cchText is an optional maximum length of text\$. Defaults to 255

count = [CONTROLCMD](#)(window | dialog, ID, @LVGETSELCOUNT)

Use this function to retrieve the number of selected items

count = [CONTROLCMD](#)(window | dialog, ID, @LVGETCOUNT)

Use this function to retrieve the total number of items in the list view control

[CONTROLCMD](#) window | dialog, ID, [@LVSETCOLUMNTEXT](#), column, text\$

Use this statement to change the text of a column in report view.

column is the zero-based index of the column to change.

text\$ is the new column text

[CONTROLCMD](#)(window | dialog, ID, [@LVGETCOLUMNTEXT](#), column, text\$ {, cchText})

Use this function to retrieve the text of a column.

column is the zero-based index of the column to retrieve

text\$ is a variable of type STRING to store retrieved text.

cchText is an optional maximum length of text\$. Defaults to 255.

[CONTROLCMD](#) window | dialog, ID, [@LVSETCOLWIDTH](#), column, width

Use this function to set a columns width

column is the zero-based index of the column to change.

width is the new width of the column in pixels

width = [CONTROLCMD](#)(window | dialog, ID, [@LVGETCOLWIDTH](#), column)

Use this function to retrieve a columns width

column is the zero-based index of the column to change.

[CONTROLCMD](#) window | dialog, ID, [@LVSETSELECTED](#), item

selected = [CONTROLCMD](#)(window | dialog, ID, [@LVGETSELECTED](#), item)

Use this function to determine the selected state of the item.

item is the zero-based index of the item.

position = [CONTROLCMD](#)(window | dialog, ID, [@LVFINDITEM](#), text\$)

Searches for an item. Returns the zero-based index of the item or -1 if the item could not be found.

text\$ is the case sensitive string to search for.

position = [CONTROLCMD](#)(window | dialog, ID, [@LVGETTOPINDEX](#))

Use this function to retrieve the zero-based index of the first item visible in the control

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

`text$ = GETCONTROLTEXT (window | dialog, ID)`
Used to retrieve the hidden caption of the control.

`GETSIZE(window | dialog, L, T, W, H,, ID)`
Gets the size of a control.

`MODIFYEXSTYLE(window | dialog, add , remove, ID)`
Adds or removes extended styles from a control.

`MODIFYSTYLE(window | dialog, add, remove, ID)`
Adds or removes styles from a control.

`REDRAWFRAME(window | dialog, ID)`
Redraws the control after the style has been modified.

`SENDMESSAGE window | dialog, msg, wparam, lparam , ID`
Sends a message to the control for advanced functionality.

`SETCONTROLCOLOR window | dialog, ID, fg, bg`
Sets the text color and background color of the control.

`SETCONTROLTEXT window | dialog, ID, text`
Used to change the hidden caption of the control.

`SETFOCUS window | dialog, ID`
Gives the control the input focus.

`SETFONT window | dialog, typeface, height, weight , flags , ID`
Changes the font of all text in the control.

`SETSIZE window | dialog, L, T, W, H , ID`
Changes the size of the control.

`SHOWWINDOW window | dialog, flags, ID`
Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

Notification messages

A list view control sends notification messages to the parent window or dialog in the `@NOTIFYCODE` variable. The ID of the control is found in `@CONTROLID` the same as standard controls. The following notification messages are supported:

@NMCLICK

User has left clicked in the control

@NMDBLCLK

User has double clicked in the control

@NMKILLFOCUS

The control has lost the input focus

@NMSETFOCUS

The control has received the input focus

@NMRCLICK

User has right clicked in the control

@LVNCOLUMNCLICK

Indicates that the user clicked a column header in report view.

@QUAL contains a memory handle to a **NMLISTVIEW** data type.

@LVNKEYDOWN

Signals a keyboard event

@QUAL contains a memory handle to a **LVKEYDOWN** data type

@LVNBEGINLABELEDIT

Signals the start of in-place label editing

@LVNENDLABELEDIT

Signals the end of label editing

@LVNITEMCHANGED

Indicates that an item has changed.

@QUAL contains a memory handle to a **NMLISTVIEW** data type.

@LVNITEMCHANGING

Indicates that an item is in the process of changing

@QUAL contains a memory handle to a **NMLISTVIEW** data type.

@LVNINSERTITEM

Signals the insertion of a new list view item.

@QUAL contains a memory handle to a **NMLISTVIEW** data type.

@LVNDELETEITEM

Signals the deletion of a specific item

@QUAL contains a memory handle to a **NMLISTVIEW** data type.

Data types

Some notification messages set @LPARAM to a UDT pointer. The following UDT's are commonly used with list view notifications.

```
TYPE NMLISTVIEW
  def hwndFrom:UINT
  def idFrom:INT
  def code:INT
  def iItem:INT
  def iSubItem:INT
  def uNewState:UINT
  def uOldState:UINT
  def uChanged:UINT
  def ptActionx:INT
  def ptActiony:INT
  def lParam:INT
ENDTYPE
```

```
TYPE LVKEYDOWN
  def hwndFrom:UINT
  def idFrom:INT
  def code:INT
  def vkey:WORD
  def flags:INT
ENDTYPE
```

Reading data types

To read the UDT pointer stored in @LPARAM use pointer dereferencing and type casting

Example fragment:

```
'standard NM_LISTVIEW
TYPE NMLISTVIEW
  def hwndFrom:UINT
  def idFrom:INT
  def code:INT
  def iItem:INT
  def iSubItem:INT
  def uNewState:UINT
  def uOldState:UINT
  def uChanged:UINT
  def ptActionx:INT
  def ptActiony:INT
  def lParam:INT
ENDTYPE

...
SUB Handler( ),INT
SELECT @MESSAGE
  CASE @IDCONTROL
    IF(@NOTIFYCODE = @LVNCOLUMNCLICK)
      CONTROLCMD d1,1,@LVSETCOLUMNTEXT,*<NMLISTVIEW>@LPARAM.iSubItem,"Clicked!"
    ENDIF
  ENDSELECT
RETURN 0
```

ENDSUB

See the sample file *listview.iwb* for a demonstration of reading a UDT handle from @LPARAM

10.12.1!Pager Controls - WIP

About Pager controls

A *pager control* is a window container that is used with a window that does not have enough display area to show all of its content.

Creating the control

UINT = PagerControl(win as WINDOW,l as INT,t as INT,w as INT,h as INT,flags as INT,exStyle as INT,id as UINT)

Pager control styles

@PGS_VERT

Creates a pager control that can be scrolled vertically. This is the default.

@PGS_HORZ

Creates a pager control that can be scrolled horizontally. This style and the @PGS_VERT style are mutually exclusive and cannot be combined.

@PGS_AUTOSCROLL

The pager control will scroll when the user hovers the mouse over one of the scroll buttons.

@PGS_DRAGNDROP

The contained window can be a drag-and-drop target. The pager control will automatically scroll if an item is dragged from outside the pager over one of the scroll buttons.

Pager control functions and statements

[pcForwardMouse](#)(window | dialog, ID, bForward)

Enables or disables mouse forwarding for the pager control.

INT = [pcGetButtonState](#)(window | dialog, ID, button)

Retrieves the state of the specified button in a pager control.

INT = [pcGetPos](#)(window | dialog, ID)

Retrieves the current scroll position of the pager control.

[pcRecalcSize](#)(window | dialog, ID)

Forces the pager control to recalculate the size of the contained window.

[pcSetBackColor](#)(window | dialog, ID, clr)

Sets the current background color for the pager control.

[pcSetBorderSize](#)(window | dialog, ID, size)

Sets the current border size for the pager control.

[pcSetButtonSize](#)(window | dialog, ID, size)

Sets the current button size for the pager control.

[pcSetChild](#)(window | dialog, ID, child)

Sets the contained control for the pager control.

[pcSetChildHwnd](#)(window | dialog, ID, hwndChild)

Sets the contained window for the pager control. This message will not change the parent of the contained window; it only assigns a window handle to the pager control for scrolling.

[pcSetPos](#)(window | dialog, ID, pos)

Sets the current scroll position of the pager control.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam, ID
Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg
Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text
Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID
Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight, flags, ID
The text printed in a control will use the default font specified in the display control panel unless changed with the [SETFONT](#) statement.

[SETSIZE](#) window | dialog, L, T, W, H, ID
Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID
Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

Notification messages

10.12.1 Progress Controls - WIP

About Progress controls

A progress bar is a window that an application can use to indicate the progress of a lengthy operation

Creating the control

UINT = ProgressControl(win as WINDOW, l as INT, t as INT, w as INT, h as INT, flags as INT, exStyle as INT, id as UINT)

Progress control styles

@PBS_SMOOTH

The progress bar displays progress status in a smooth scrolling bar instead of the default segmented bar. This style is supported only in the Windows Classic theme.

@PBS_VERTICAL

The progress bar displays progress status vertically, from bottom to top.

@PBS_MARQUEE

The progress indicator does not grow in size but instead moves repeatedly along the length of the bar, indicating activity without specifying what proportion of the progress is complete.

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

Progress control functions and statements

INT = [GetProgressPosition](#)(window | dialog, ID)

Retrieves the current position of the progress bar.

[ProgressStepIt](#)(window | dialog, ID)

Advances the current position for a progress bar by the step increment and redraws the bar to reflect the new position.

[SetProgressBarColor](#)(window | dialog, ID, clr)

Sets the color of the progress indicator bar in the progress bar control.

[SetProgressDelta](#)(window | dialog, ID, delta)

Advances the current position of a progress bar by a specified increment and redraws the bar to reflect the new position.

[SetProgressMarquee](#)(window | dialog, ID, bEnable, time)

Sets the progress bar to marquee mode. This causes the progress bar to move like a marquee.

[SetProgressPosition](#)(window | dialog, ID, pos)

Sets the current position for a progress bar and redraws the bar to reflect the new position.

[SetProgressRange](#)(window | dialog, ID, min, max)

Sets the minimum and maximum values for a progress bar and redraws the bar to reflect the new range.

[SetProgressStep](#)(window | dialog, ID, value)

Specifies the step increment for a progress bar.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wParam, lParam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight , flags , ID

The text printed in a control will use the default font specified in the display control panel unless changed with the [SETFONT](#) statement.

[SETSIZE](#) window | dialog, L, T, W, H , ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

Notification messages

10.12.17 RadioButton Controls - WIP

About RadioButton controls

Creating the control

Radiobutton controls are created either through the [Form Editor](#) or manually with the [CONTROL](#) statement.

RadioButton control styles

The following RadioButton style flags can be specified in the [CONTROL](#) statement or by ticking the corresponding check box in the control properties dialog of the [Form Editor](#):

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the

next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSTEMMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

```
if getstate( t,1039 ) then style+="@CTLBTNFLAT|"
```

```
if getstate( t,1040 ) then style+="@LEFTTEXT|"
```

@SS_NOTIFY

Causes the static control to send @BN_CLICKED and @BN_DBLCLK messages to the parent window/dialog.

RadioButton control functions and statements

[SETSTATE](#) window | dialog, ID, state

Sets or resets a checkbox or radio button control. State can either be 0 to uncheck the control or 1 to check the control.

state = [GETSTATE](#) (window | dialog, ID)

Returns the state of a checkbox or radio button control. Returns 1 if control is checked and 0 if control is unchecked. Radio buttons in a group are mutually exclusive. When a radio button is selected your program will receive an @IDCONTROL message with @CONTROLID containing the newly selected radio button. If the radio button is created outside of a group, you will need to use GETSTATE to determine whether the button is selected.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control. If there is a manifest file the text color does not change.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight , flags , ID

Changes the font of all text in the control.

[SETSIZE](#) window | dialog, L, T, W, H , ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

Notification messages

@BN_CLICKED

The control has been clicked. Sent to the parent provided the `@SS_NOTIFY` style was used when the control was created.

@BN_DBLCLK

The control has been double clicked. Sent to the parent provided the `@SS_NOTIFY` style was used when the control was created.

10.12.1 Rebar Controls - WIP

About Rebar controls

Rebar controls act as containers for child windows.

Creating the control

```
UINT = RebarControl(win as WINDOW, flags as INT, exStyle as INT, id as UINT)
```

Rebar control styles

@RBS_TOOLTIPS

Not yet supported.

@RBS_VARHEIGHT

The rebar control displays bands at the minimum required height, when possible.

@RBS_BANDBORDERS

The rebar control displays narrow lines to separate adjacent bands.

@RBS_FIXEDORDER

The rebar control always displays bands in the same order.

@RBS_REGISTERDROP

The rebar control generates @RBN_GETOBJECT notification messages when an object is dragged over a band in the control.

@RBS_AUTOSIZE

The rebar control will automatically change the layout of the bands when the size or position of the control changes. An @RBN_AUTOSIZE notification will be sent when this occurs.

@RBS_VERTICALGRIPPER

This always has the vertical gripper (default for horizontal mode)

@RBS_DBLCLKTOGGLE

The rebar band will toggle its maximized or minimized state when the user double-clicks the band.

Rebar control functions and statements

[rbAddBand](#)(window | dialog, ID, index, style, cx, cxMinChild, cyMinChild)

Inserts a new band in the rebar control.

[rbSetBandBitmap](#)(window | dialog, ID, index, hBitmap)

Sets the background bitmap used for a band.

[rbSetBandChild](#)(window | dialog, ID, index, child)

Sets the child control contained by the band in a rebar control.

[rbSetBandChildHandle](#)(window | dialog, ID, index, hwndChild)

Sets the child window contained by the band in a rebar control.

[rbSetBandColors](#)(window | dialog, ID, index, fore, back)

Sets the foreground and background color of a band.

[rbSetBandText](#)(window | dialog, ID, index, text)

Sets the text for a band in a rebar control.

[rbShowBand](#)(window | dialog, ID, index, bShow)

Shows or hides a given band in a rebar control.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam, ID
Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg
Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text
Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID
Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight, flags, ID
The text printed in a control will use the default font specified in the display control panel unless changed with the [SETFONT](#) statement.

[SETSIZE](#) window | dialog, L, T, W, H, ID
Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID
Changes the visibility state of the control. Use [@SWHIDE](#) to hide the control and [@SWRESTORE](#) to show the control

Notification messages

10.12.1 RichEdit Controls - WIP

About rich edit controls

A "rich edit control" is a window in which the user can enter and edit text. The text can be assigned character and line formatting, and can include embedded OLE objects. Rich edit controls provide a programming interface for formatting text. However, an application must implement any user interface components necessary to make formatting operations available to the user.

Rich edit controls support almost all of the commands and notification messages used with multiline

edit controls. Thus, applications that already use edit controls can be easily changed to use rich edit controls. Additional commands and notifications enable applications to access the functionality unique to rich edit controls.

Rich edit controls can save and load text in either straight ASCII format or in rich text format (*.rtf)

Creating the control

Rich edit controls are created in the same manner as the standard control types, either through the [Form Editor](#) or manually with the [CONTROL](#) statement. Rich edit controls support the same creation flags as standard multiline edit controls except for the @CTLEDITPASS style. A rich edit control can also be created as a single line edit control.

Rich edit control styles

The following rich edit style flags can be specified in the [CONTROL](#) statement or by ticking the corresponding check box in the control properties of the [Form Editor](#):

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSTEMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

@CTEDITLEFT

Text is left justified in the edit control

@CTEDITRIGHT

Text is right justified in the edit control

@CTEDITMULTI

Designates a multiline edit control. The default is single-line edit control. When the multiline edit control is in a dialog box, the default response to pressing the ENTER key is to activate the default button. To use the ENTER key as a carriage return, use the @CTEDITRETURN style. When the

multiline edit control is not in a dialog box and the `@CTEDITAUTOV` style is specified, the edit control shows as many lines as possible and scrolls vertically when the user presses the ENTER key. If you do not specify `@CTEDITAUTOV`, the edit control shows as many lines as possible and beeps if the user presses the ENTER key when no more lines can be displayed. If you specify the `@CTEDITAUTOH` style, the multiline edit control automatically scrolls horizontally when the caret goes past the right edge of the control. To start a new line, the user must press the ENTER key. If you do not specify `@CTEDITAUTOH`, the control automatically wraps words to the beginning of the next line when necessary. A new line is also started if the user presses the ENTER key. The window size determines the position of the word wrap. If the window size changes, the word wrapping position changes and the text is redisplayed. Multiline edit controls can have scroll bars. An edit control with scroll bars processes its own scroll bar messages. Note that edit controls without scroll bars scroll as described in the previous paragraphs and process any scroll messages sent by the parent window.

@CTEDITCENTER

Text is centered within the edit control

@CTEDITRO

The edit control is read only and text can be displayed but not entered

@CTEDITAUTOH

Automatically scrolls text to the right by 10 characters when the user types a character at the end of the line. When the user presses the ENTER key, the control scrolls all text back to position zero.

@CTEDITAUTOV

Automatically scrolls text up one page when the user presses the ENTER key on the last line.

@CTEDITRETURN

Specifies that a carriage return be inserted when the user presses the ENTER key while entering text into a multiline edit control in a dialog box. If you do not specify this style, pressing the ENTER key has the same effect as pressing the dialog box's default push button. This style has no effect on a single-line edit control.

@CTEDITNUMBER

Restricts text entered in a single line edit control to numerals only (0 - 9)

In addition to the edit styles the following window styles can also be specified:

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the `@GROUP` style after the first control

belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@HSCROLL

The control has a horizontal scroll bar

@VSCROLL

The control has a vertical scrollbar

Controlling the rich edit control.

The rich edit control is accessed through the [CONTROLCMD](#) statement/function. [CONTROLCMD](#) can be used as a statement or function depending on the command issued.

Clipboard operations

[CONTROLCMD](#) window | dialog, ID, **@RTCUT**

Use this command to delete (cut) the current selection (if any) in the rich edit control and copy the deleted text to the Clipboard.

[CONTROLCMD](#) window | dialog, ID, **@RTCOPY**

Use this command to copy the current selection (if any) in the rich edit control to the Clipboard.

[CONTROLCMD](#) window | dialog, ID, **@RTPASTE**

Use this command to insert the data from the Clipboard into the rich edit control at the insertion point, the location of the caret. Data is inserted only if the Clipboard contains data in a recognized format.

Line operations

count = [CONTROLCMD](#) (window | dialog, ID, **@RTGETLINECOUNT**)

Use this function to retrieve the number of lines in the rich edit control

[CONTROLCMD](#) (window | dialog, ID, **@RTGETLINE**, linenum, line\$ {,cchLine})

Use this function to retrieve a line of text from the rich edit control.

Linenum is the 0 based index of the line to retrieve, *cchLine* is the size of the *line\$* STRING variable, defaults at 255.

line = [CONTROLCMD](#) (window | dialog, ID, **@RTGETFIRSTLINE**)

Use this function to retrieve the zero-based index of the uppermost visible line.

line = [CONTROLCMD](#) (window | dialog, ID, **@RTLINEFROMCHAR**, index)

Use this function to retrieve the line number of the line that contains the specified character index. Index is the number of characters from the beginning of the rich edit control. For character counting, an OLE item is counted as a single character

char_index = [CONTROLCMD](#) (window | dialog, ID, **@RTCHARFROMLINE**, linenum)

Use this function to retrieve the character index of the first character of the specified line.

length = [CONTROLCMD](#) (window | dialog, ID, [@RTGETLINELENGTH](#), index)

Use this function to retrieve the length of a line in a rich edit control. When

[@RTGETLINELENGTH](#) is called for a multiple-line edit control, the return value is the length (in bytes) of the line specified by index. When [@RTGETLINELENGTH](#) is called for a single-line edit control, the return value is the length (in bytes) of the text in the edit control. *Index* specifies the character index of a character in the line whose length is to be retrieved. If this parameter is -1, the length of the current line (the line that contains the caret) is returned

[CONTROLCMD](#) window | dialog, ID, [@RTSCROLL](#), lines, chars

Use this command to scroll the text of a multiple-line edit control. The rich edit control does not scroll vertically past the last line of text in the edit control. If the current line plus the number of lines specified by *lines* exceeds the total number of lines in the edit control, the value is adjusted so that the last line of the edit control is scrolled to the top of the edit-control window.

Selection operations

[CONTROLCMD](#) window | dialog, ID, [@RTGETSELECTION](#), varStart, varEnd

Use this command to retrieve the current selection of the rich edit control.

varStart and *varEnd* must be of type INT. The zero-based index of the first and last characters selected are copied into the two variables. The selection includes everything if *varStart* = 0 and *varEnd* = -1.

[CONTROLCMD](#) (window | dialog, ID, [@RTGETSELTEXT](#), text\$ {,cchText})

Use this function to retrieve the text of the selection. *cchText* is the size of the *text\$* variable, defaults to 255

[CONTROLCMD](#) window | dialog, ID, [@RTDELETESEL](#)

Use this statement to delete the current selection. The deletion performed by [@RTDELETESEL](#) can be undone by using [@RTUNDO](#)

[CONTROLCMD](#) window | dialog, ID, [@RTSETSELECTION](#), start, end

Use this statement to set the current selection in the rich edit control

start and *end* are the zero-based character indexes of the selection. If *start* = 0 and *end* = -1 then all of the text is selected.

[CONTROLCMD](#) window | dialog, ID, [@RTREPLACESEL](#), text\$

Use this statement to replace the current selection text

[CONTROLCMD](#) window | dialog, ID, [@RTHIDSEL](#), hide

Use this statement to change the visibility of the current selection. If *hide* = 1 then the selection is hidden. If *hide* = 0 then the selection is shown.

Formatting operations

[CONTROLCMD](#) window | dialog, ID, [@RTSETDEFAULTFONT](#), name\$, height, bold,

effects

Changes the default font in the rich edit control. The default font is used when no other character formatting has been specified.

Height is specified in points. If a bold font is required set bold = 1. Effects can be a combination of @SFITALIC, @SFUNDERLINE, or @SFSTRIKEOUT. Combine effects with the '|' operator.

CONTROLCMD window | dialog, ID, @RTSETSELFONT, name\$, height, bold, effects

Changes the font of the currently selected text in the rich edit control. If there is no selection the font is changed for all characters entered after the insertion point.

Height is specified in points. If a bold font is required set bold = 1. Effects can be a combination of @SFITALIC, @SFUNDERLINE, or @SFSTRIKEOUT. Combine effects with the '|' operator.

CONTROLCMD window | dialog, ID, @RTSETDEFAULTCOLOR, color

Changes the default text color of the rich edit control. The default text color is used when no other character formatting has been specified.

Color is a color value created with the RGB function.

CONTROLCMD window | dialog, ID, @RTSETSELCOLOR, color

Changes the selection text color of the rich edit control. If there is no selection the color is changed for all characters entered after the insertion point.

Color is a color value created with the RGB function.

CONTROLCMD window | dialog, ID, @RTSETALIGNMENT, alignment

Changes the alignment of the current selection. If there is no selection the alignment is set for the insertion point.

Valid alignment values are @RTALIGNLEFT, @RTALIGNCENTER and @RTALIGNRIGHT

CONTROLCMD window | dialog, ID, @RTSETCHAROFFSET, offset

Character offset, in twips, from the baseline. If offset is positive, the character is a superscript; if it is negative, the character is a subscript. There are 1440 twips in an inch, 20 twips in a point.

Editing operations

CONTROLCMD window | dialog, ID, @RTUNDO

Use this statement to undo the last editing operation. An undo operation can also be undone. For example, you can restore deleted text with the first call to Undo. As long as there is no intervening edit operation, you can remove the text again with a second call to Undo.

return = **CONTROLCMD** (window | dialog, ID, @RTCANUNDO)

Use this function to determine if the last editing operation can be undone. Returns 0 if the last operation cannot be undone.

CONTROLCMD window | dialog, ID, @RTEMPTYUNDO

Use this statement to reset (clear) the undo flag of this rich edit control. The control will now be unable to undo the last editing operation. The undo flag is set whenever an operation within the rich edit control can be undone.

return = [CONTROLCMD](#) (window | dialog, ID, **@RTSAVE**, FILE | STRING, type)

Use this function to save the contents of the rich edit control to an open file or into a string variable. Type is either 0 to store the contents in plain ASCII format or 1 to store the contents in rich text format (*.rtf). Returns the number of characters saved. FILE is a variable of type FILE and must have been successfully opened for writing with the OPENFILE function.

return = [CONTROLCMD](#) (window | dialog, ID, **@RTLOAD**, FILE | STRING, type)

Use this function to load the contents of the rich edit control from an open file or a string variable. Type is either 0 to load the contents in plain ASCII format or 1 to load the contents in rich text format (*.rtf). Returns number of characters loaded. FILE is a variable of type FILE and must have been successfully opened for reading with the OPENFILE function.

General operations

return = [CONTROLCMD](#) (window | dialog, ID, **@RTGETMODIFIED**)

Use this function to determine if the contents of the rich edit control have changed. Returns 1 if the contents have been modified, 0 otherwise.

[CONTROLCMD](#) window | dialog, ID, **@RTSETMODIFIED**, mod

Sets the modified flag of the rich edit control.

Mod can be 0 to reset the flag or 1 to set it.

pos = [CONTROLCMD](#) (window | dialog, ID, **@RTFINDTEXT**, text\$, start_pos, ignore_case)

Use this function to find text within the rich edit control. Returns the zero-based index of the first character in the control that matches the string in text\$. Returns -1 if there are no matches. start_pos specifies a starting character index to begin the search. If ignore_case = 0 then the function performs a case sensitive search, if ignore_case = 1 then the case of the search string is ignored.

length = [CONTROLCMD](#) (window | dialog, ID, **@RTGETLIMITTEXT**)

Use this function to get the text limit for this rich edit control. The text limit is the maximum amount of text, in bytes, the rich edit control can accept either through pasting, typing or with the @RTLOAD function.

[CONTROLCMD](#) window | dialog, ID, **@RTSETLIMITTEXT**, length

Use this function to set the text limit for this rich edit control. The text limit is the maximum amount of text, in bytes, the rich edit control can accept either through pasting, typing or with the @RTLOAD function. The default limit is 32767 bytes.

[CONTROLCMD](#) window | dialog, ID, **@RTSETLINEWIDTH**, width

Use this statement to set the line width for word wrapping in the rich edit control. Width is specified in twips. There are 1440 twips in 1 inch.

[CONTROLCMD](#) window | dialog, ID, **@RTSETMARGINS**, left, right

Use this statement to set the visible left and right margins of the rich edit control.

Left and right are specified in pixels.

length = [CONTROLCMD](#) (window | dialog, ID, @RTGETTEXTLENGTH)

Returns the text length, in bytes, of the rich edit control.

[CONTROLCMD](#) window | dialog, ID, @RTPRINT {, margin}

Opens the standard print dialog and prints the contents of the rich edit control. Optional margin value specifies the printer margin in twips. 1440 twips = 1 inch. Default margin is 1/2 inch.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the text of the control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change all the text of the control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

SETFONT window | dialog, typeface, height, weight , flags , ID
Changes the font of all text in the control.

SETSIZE window | dialog, L, T, W, H , ID
Changes the size of the control.

SHOWWINDOW window | dialog, flags, ID
Changes the visibility state of the control. Use **@SWHIDE** to hide the control and **@SWRESTORE** to show the control

Notification event control

mask = **CONTROLCMD** (window | dialog, ID, **@RTGETEVENTMASK**)
Returns the event mask of the rich edit control.

CONTROLCMD window | dialog, ID, **@RTSETEVENTMASK**, mask
Sets the notification event mask for the rich edit control. The event mask controls which notification messages are sent to the parent window or dialog in the **@NOTIFYCODE** variable. Multiple events can be specified by using the | operator The following mask values are available:

@ENMNONE

Only the standard events **@ENKILLFOCUS**, **@ENSETFOCUS**, **@ENMAXTEXT** and **@ENERRSPACE** are sent.

@ENMCHANGE

The control sends **@ENCHANGE** events

@ENMUPDATE

The control sends **@ENUPDATE** events

@ENMSCROLL

The controls sends **@ENHSCROLL** and **@ENVSCROLL** events

@ENMREQUESTRESIZE

The control sends **@ENREQUESTRESIZE** events

@ENMSELCHANGE

The control sends **@ENSELCHANGE** events

Notification messages

A rich edit controls sends notification messages to the parent window or dialog in the **@NOTIFYCODE** variable. The ID of the control is found in **@CONTROLID** the same as standard controls. The following notification messages are supported:

@ENKILLFOCUS

Control has lost input focus

@ENSETFOCUS

Control has been given the input focus

@ENERRSPACE

Control could not complete an operation because there was not enough memory available

@ENMAXTEXT

The limit set by @RTSETLIMITTEXT has been reached,

@ENUPDATE

The contents of the control are about to change
Must be enabled with @RTSETEVENTMASK

@ENSELCHANGE

The current selection has changed.
Must be enabled with @RTSETEVENTMASK

@ENCHANGE

The contents of the control have changed.
Must be enabled with @RTSETEVENTMASK

@ENHSCROLL

The user has clicked the edit control's horizontal scroll bar. Windows sends this message before updating the control.
Must be enabled with @RTSETEVENTMASK

@ENVSCROLL

The user has clicked the edit control's vertical scroll bar. Windows sends this message before updating the control.
Must be enabled with @RTSETEVENTMASK

@ENREQUESTRESIZE

The control's contents are either smaller or larger than the control's window size.
Must be enabled with @RTSETEVENTMASK

Miscellaneous

The rich edit control also supports the following IW BASIC statements and functions:

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the background color of the rich edit control. The foreground color is ignored as the rich edit controls supports individual character formatting.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the rich edit control. If the rich edit control is disabled no text can be entered and the control will not receive input focus when clicked on.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus. The rich edit control shows the caret.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam ,ID

Sends a message to the control for advanced functionality.

[SHOWWINDOW](#) window, flags, ID

Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

[SETSIZE](#) dialog|window, L, T, W, H ,ID

Changes the size of the control. The rich edit control is redrawn and the text will be formatted to match the new size of the control. The dimensions include the borders, if any, of the control.

10.12.2(Scrollbar Controls - WIP

About scroll bar controls

A scroll bar consists of a shaded shaft with an arrow button at each end and a *scroll box* (sometimes called a thumb) between the arrow buttons. A scroll bar represents the overall length or width of a data object in a window's client area; the scroll box represents the portion of the object that is visible in the client area. The position of the scroll box changes whenever the user scrolls a data object to display a different portion of it. Windows also adjusts the size of a scroll bar's scroll box so that it indicates what portion of the entire data object is currently visible in the window. If most of the object is visible, the scroll box occupies most of the scroll bar's shaft. Similarly, if only a small portion of the object is visible, the scroll box occupies a small part of the shaft.

A *scroll bar control* is a control that is separate from the window frame and can be placed anywhere in a dialog or window. A scroll bar control appears and functions like a standard scroll bar, but it is a separate window. As a separate window, a scroll bar control receives direct input focus, indicated by a flashing caret displayed in the scroll box. Unlike a standard scroll bar, a scroll bar control also has a built-in keyboard interface that enables the user to direct scrolling. You can use as many scroll bar controls as needed in a single window. When you create a scroll bar control, you must specify the scroll bar's size and position. However, if a scroll bar control's window can be resized, adjustments to the scroll bar's size must be made whenever the size of the window changes.

Creating the control

Scroll bar controls are created through the [Form Editor](#), manually with the [CONTROL](#) statement or by specifying [@HSCROLL](#) and [@VSCROLL](#) when creating a window.

Scroll bar control styles

The following scroll bar style flags can be specified in the [CONTROL](#) statement or by ticking the corresponding check box in the control properties of the [Form Editor](#):

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the [@GROUP](#) style after the first control belong to the same group. The next control with the [@GROUP](#) style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSTEMMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

@CTSCROLLHORIZ

Creates a horizontal scroll bar. Automatically set when using the [Form Editor](#).

@CTSCROLLVERT

Creates a vertical scroll bar. Automatically set when using the [Form Editor](#).

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the [@GROUP](#) style after the first control belong to the same group. The next control with the [@GROUP](#) style starts the next group (that is, one group ends where the next begins)

Scroll bar control functions and statements

The following IW BASIC functions and statements are used to communicate with the scroll bar control.

[SETSCROLLRANGE](#) window | dialog, ID, min, max

Sets the minimum and maximum range of a scrollbar control to min and max. All values returned by the scrollbar will be between min and max. If ID = -1 then sets the range of the windows horizontal scrollbar. If ID = -2 then sets the range of the windows vertical scrollbar. ID must be a scrollbar control.

[GETSCROLLRANGE](#) window | dialog, ID, varMin, varMax

Stores the scrollbars range into the variables specified by varMin and varMax. The variables must be of type INT. If ID = -1 then stores the range of the windows horizontal scrollbar. If ID = -2 then stores the range of the windows vertical scrollbar. ID must be a scrollbar control. :

[SETSCROLLPOS](#) window | dialog, ID, position

Sets the slider position of a scrollbar. If ID = -1 then sets the scroll position of the windows horizontal scrollbar. If ID = -2 then sets the scroll position of the windows vertical scrollbar. Any other ID value is a user-defined scrollbar. Position must be between the minimum and maximum values set by the SETSCROLLRANGE statement.

position = [GETSCROLLPOS](#) (window | dialog, ID)

Returns the slider position of a scrollbar. If ID = -1 then returns the scroll position of the windows horizontal scrollbar. If ID = -2 then returns the scroll position of the windows vertical scrollbar. Any other ID value is a user-defined scrollbar.

position = [GETTHUMBPOS](#)(window | dialog, ID)

Returns the current thumb track position of a scrollbar. If ID = -1 then returns the thumb track position of the windows horizontal scrollbar. If ID = -2 then returns the thumb track position of the windows vertical scrollbar. Any other ID value is a user-defined scrollbar.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the scrollbar control. If the scrollbar control is disabled no scrolling can be performed and the control will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

GETCONTROLTEXT is used to retrieve the hidden caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a window, dialog or control. The dimensions returned include the borders and caption.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a window, dialog or control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a window, dialog or control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the border, caption and non-client areas of a window, dialog or control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the scrollbar control.

[SETCONTROLTEXT](#) window | dialog, ID, text

SETCONTROLTEXT is used to change the hidden caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus. The scrollbar control shows a highlight rectangle in the thumb slider.

[SETFONT](#) window | dialog, typeface, height, weight , flags , ID

The text printed in a control will use the default font specified in the display control panel unless changed with the [SETFONT](#) statement.

Height and weight can both be 0 in which case a default size and weight will be used. Weight ranges from 0 to 1000 with 700 being standard for bold fonts and 400 for normal fonts. Flags can be a combination of [@SFITALIC](#), [@SFUNDERLINE](#) or [@SFSTRIKEOUT](#) for italicized and underlined fonts. If an ID is specified then the font of a control is changed.

The height parameter is specified in points. 1 point is equal to 1/72nd of an inch. If you want a font that is 1/2 an inch high you would specify a point size of 36.

[SETSIZE](#) window | dialog, L, T, W, H , ID

Changes the size of the control. The scrollbar control is redrawn and the text will be formatted to match the new size of the control. The dimensions include the borders of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use [@SWHIDE](#) to hide the control and [@SWRESTORE](#) to show the control

Messages

Windows sends two messages to indicate the user has performed an action with the scrollbar.

@IDHSCROLL

This message is sent whenever an action has been performed with either the windows horizontal scroll bar or a horizontal scroll bar created with the CONTROL statement. @CONTROLID will contain the ID of the scroll bar control or 0 for a windows horizontal scroll bar.

@IDVSCROLL

This message is sent whenever an action has been performed with either the windows vertical scroll bar or a vertical scroll bar created with the CONTROL statement. @CONTROLID will contain the ID of the scroll bar control or 0 for a windows vertical scroll bar.

Notification messages

Unlike other controls, a scroll bar returns notification messages in @WPARAM instead of @NOTIFYCODE. Notification messages for scroll bars are sent to inform the program that the user *wants* to perform an action. It is up to the program to scroll the data and position the scroll box (thumb track) of the scroll bar to the correct position,

@SBLEFT

Scroll to the far left. Sent when the user drags the scroll box to the far left

@SBENDSCROLL

End scroll.

@SBLINELEFT

The user clicked the left scroll arrow.

@SBLINERIGHT

The user clicked the right scroll arrow.

@SBPAGELEFT

Scroll one page left

@SBPAGERIGHT

Scroll one page right

@SBRIGHT

Scroll to the far right. Sent when the user drags the scroll box to the far right

@SBTHUMBPOS

Scroll to absolute position. Use GETTHUMBPOS for the position

@SBTHUMBTRACK

Drag scroll box to a position. Use GETTHUMBPOS for the position

@SBBOTTOM

Scroll to the bottom. Sent when the user drags the scroll bar to the bottom.

@SBLINEDOWN

The user clicks the bottom scroll arrow.

@SBLINEUP

The user clicked the top scroll arrow.

@SBPAGEDOWN

Scroll one page down

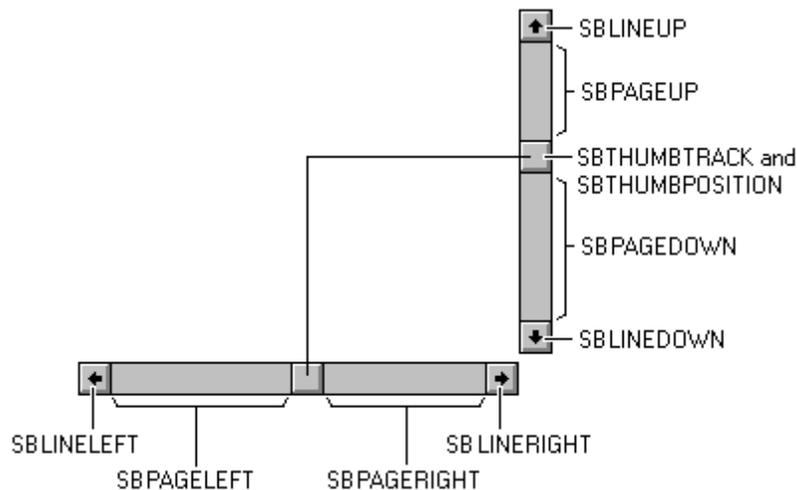
@SBPAGEUP

Scroll one page up

@SBTOP

Scroll to the top. Sent when the user drags the scroll box to the top.

The following illustration shows the relationships between the notification codes and the parts of the scroll bars



10.12.2 Spinner Controls - WIP

About Spinner controls

An up-down control is a pair of arrow buttons that the user can click to increment or decrement a value, such as a scroll position or a number displayed in a companion control (called a buddy window).

Creating the control

```
UINT = SpinnerControl(win as WINDOW, l as INT, t as INT, w as INT, h as INT, flags as INT,
exStyle as INT, id as UINT)
```

Spinner control styles

@UDS_WRAP

Causes the position to "wrap" if it is incremented or decremented beyond the ending or beginning of the range.

@UDS_SETBUDDYINT

Automatically changes the text of the buddy.

@UDS_ALIGNRIGHT

Positions the up-down control next to the right edge of the buddy window. The width of the buddy window is decreased to accommodate the width of the up-down control.

@UDS_ALIGNLEFT

Positions the up-down control next to the left edge of the buddy window. The buddy window is moved to the right, and its width is decreased to accommodate the width of the up-down control.

@UDS_AUTOBUDDY

Automatically chooses the previous created control as its buddy.

@UDS_ARROWKEYS

Causes the up-down control to increment and decrement the position when the UP ARROW and DOWN ARROW keys are pressed.

@UDS_HORZ

Causes the up-down control's arrows to point left and right instead of up and down.

@UDS_NOTHOUSANDS

Does not insert a thousands separator between every three decimal digits.

@UDS_HOTTRACK

Causes the control to exhibit "hot tracking" behavior.

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for

controls created in a window.

Spinner control functions and statements

INT = [GetSpinnerBase](#)(window | dialog, ID)

Retrieves the current radix base for the spin button control.

UINT = [GetSpinnerBuddy](#)(window | dialog, ID)

Returns the id to the current buddy control for the spin button control.

INT = [GetSpinnerPosition](#)(window | dialog, ID)

Returns the position of the spin button control

INT = [GetSpinnerRangeMax](#)(window | dialog, ID)

Returns the upper limit of the range of the spinner control.

INT = [GetSpinnerRangeMin](#)(window | dialog, ID)

Returns the lower limit of the range of the spinner control.

[SetSpinnerBase](#)(window | dialog, ID, base)

Sets the current radix base for the spin button control.

[SetSpinnerBuddy](#)(window | dialog, ID, buddy)

Sets the current buddy control for the spin button control.

[SetSpinnerPosition](#)(window | dialog, ID, pos)

Sets the current position of the spin button control.

[SetSpinnerRange](#)(window | dialog, ID, min, max)

Sets the current range of the spin button control.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wParam, lParam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight , flags , ID

The text printed in a control will use the default font specified in the display control panel unless changed with the [SETFONT](#) statement.

[SETSIZE](#) window | dialog, L, T, W, H , ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use [@SWHIDE](#) to hide the control and [@SWRESTORE](#) to show the control

Notification messages

10.12.2 Static Controls - WIP

About Static controls

Buttons and static controls have the capability of displaying a bitmap instead of text. To define a bitmap button use the flag `@CTLBTNBITMAP` for a button or `@CTLSTCBITMAP` for static controls. To specify the bitmap to display set the buttons text to the complete pathname to the bitmap file with the `SETCONTROLTEXT` statement. If your button is contained in a dialog use `SETCONTROLTEXT` in response to the `@IDINITDIALOG` message.

Example of bitmap static control:

```
DEF d1 as DIALOG

CREATEDIALOG d1,0,0,295,168,0x80C80080, 0, "Bitmap Test", &dialog_main
CONTROL d1,@STATIC,"",13,14,102,102,@CTLSTCBITMAP, 1

DOMODAL d1
END

SUB dialog_main( ),INT
SELECT @MESSAGE
CASE @IDINITDIALOG
SETCONTROLTEXT d1,1,GETSTARTPATH + "bug.bmp"
CENTERWINDOW d1
ENDSELECT
RETURN 0
ENDSUB
```

Creating the control

Static controls are created either through the [Form Editor](#) or manually with the [CONTROL](#) statement.

Static control styles

The following Static style flags can be specified in the [CONTROL](#) statement or by ticking the corresponding check box in the control properties dialog of the [Form Editor](#):

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@SYSTEMMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

```
if getstate( t,1037 )=0
style+="@SS_SIMPLE|"
else
if getstate( t,1039 ) then style+="@SS_LEFT|"
if getstate( t,1040 ) then style+="@SS_CENTER|"
if getstate( t,1041 ) then style+="@SS_RIGHT|"
endif
```

if getstate(t,1038) then style+="@CTLSTCBITMAP|"

@SS_NOTIFY

Causes the static control to send @BN_CLICKED and @BN_DBLCLK messages to the parent window/dialog.

Bitmap buttons and static controls

Buttons and static controls have the capability of displaying a bitmap instead of text. To define a bitmap button use the flag @CTLBTNBITMAP for a button or @CTLSTCBITMAP for static controls. To specify the bitmap to display set the buttons text to the complete pathname to the bitmap file with the SETCONTROLTEXT statement. If your button is contained in a dialog use SETCONTROLTEXT in response to the @IDINITDIALOG message.

Example of bitmap static control:

```
DEF d1 as DIALOG

CREATEDIALOG d1,0,0,295,168,0x80C80080, 0, "Bitmap Test", &dialog_main
CONTROL d1,@STATIC,"",13,14,102,102,@CTLSTCBITMAP, 1

DOMODAL d1
END

SUB dialog_main( ),INT
SELECT @MESSAGE
CASE @IDINITDIALOG
SETCONTROLTEXT d1,1,GETSTARTPATH + "bug.bmp"
CENTERWINDOW d1
ENDSELECT
RETURN 0
ENDSUB
```

Static control functions and statements

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight , flags , ID

The text printed in a control will use the default font specified in the display control panel unless changed with the [SETFONT](#) statement.

[SETSIZE](#) window | dialog, L, T, W, H , ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use [@SWHIDE](#) to hide the control and [@SWRESTORE](#) to show the control

Notification messages

[@BN_CLICKED](#)

The control has been clicked. Sent to the parent provided the [@SS_NOTIFY](#) style was used when the control was created.

[@BN_DBLCLK](#)

The control has been double clicked. Sent to the parent provided the [@SS_NOTIFY](#) style was used when the control was created..

10.12.2: Status Controls - WIP

About status window controls

A *status window* is a horizontal window at the bottom of a parent window in which an application can display various kinds of status information. The status window can be divided into panes to display more than one type of information.

Creating the control

Status window controls are created with the [CONTROL](#) statement. The status window is created with one pane by default. The dimensions in the [CONTROL](#) statement are ignored as the status window is auto sized to the parent windows client area.

Example creation statement:

```
CONTROL win,@STATUS,"Status Window",0,0,0,0,0,2
```

When used in an MDI frame window the client area of the frame is automatically adjusted to account for the status window. For a regular dialog or window you need to take into account the height of the status bar when drawing into the window. Use the [GETSIZE](#) function to determine the height of the control.

Controlling the status window control.

The status window control is accessed through the [CONTROLCMD](#) statement/function. [CONTROLCMD](#) can be used as a statement or function depending on the command issued.

Statements and Functions

[CONTROLCMD](#) window | dialog, ID, [@SWRESIZE](#)

Use this statement to inform the status window that the parent has been resized. A status window automatically sizes itself to the parent windows client area at the bottom of the window. The typical place to use this is in response to the [@IDSIZE](#) message in the parent window.

[CONTROLCMD](#) window | dialog, ID, [@SWSETPANES](#), count, sizes[]

Sets the number of panes and the right edges of the panes.

count is the number of panes to create. sizes[] is an integer array, each element containing the pixel position of the right edge of the pane. If an element of the array is -1 then the corresponding pane extends to right edge of the client area.

[CONTROLCMD](#) window | dialog, ID, [@SWSETPANETEXT](#), pane, text\$

Use this statement to change the pane text. If there is only one pane then SETCONTROLTEXT can also be used.

pane is the zero-based index of the pane to set.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight , flags , ID

Changes the font of all text in the control.

[SETSIZE](#) window | dialog, L, T, W, H , ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

See the example file *statusbar.iwb* for a complete demonstration

10.12.2 Tab Controls - WIP

About Tab controls

A tab control is analogous to the dividers in a notebook or the labels in a file cabinet. By using a tab control, an application can define multiple pages for the same area of a window or dialog box.

Creating the control

```
UINT = TabControl(win as WINDOW,l as INT,t as INT,w as INT,h as INT,flags as INT,exStyle as INT,id as UINT)
```

Tab control styles

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the `@GROUP` style after the first control belong to the same group. The next control with the `@GROUP` style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSTEMMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

@TCS_SCROLLOPPOSITE

Unneeded tabs scroll to the opposite side of the control when a tab is selected.

@TCS_BOTTOM

Tabs appear at the bottom of the control.

@TCS_RIGHT

Tabs appear vertically on the right side of controls that use the `@TCS_VERTICAL` style.

@TCS_MULTISELECT

Multiple tabs can be selected by holding down the CTRL key when clicking.

@TCS_FLATBUTTONS

Selected tabs appear as being indented into the background.

@TCS_FORCEICONLEFT

Icons are aligned with the left edge of each fixed-width tab. This style can only be used with the `@TCS_FIXEDWIDTH` style.

@TCS_FORCELABELLEFT

Labels are aligned with the left edge of each fixed-width tab. This style can only be used with the `@TCS_FIXEDWIDTH` style, and it implies the `@TCS_FORCEICONLEFT` style.

@TCS_HOTTRACK

Items under the pointer are automatically highlighted.

@TCS_VERTICAL

Tabs appear at the left side of the control, with tab text displayed vertically.

@TCS_TABS

Tabs appear as tabs, and a border is drawn around the display area. This style is the default.

@TCS_BUTTONS

Tabs appear as buttons, and no border is drawn around the display area.

@TCS_SINGLELINE

Only one row of tabs is displayed.

@TCS_MULTILINE

Multiple rows of tabs are displayed, if necessary, so all tabs are visible at once.

@TCS_RIGHTJUSTIFY

The width of each tab is increased, if necessary, so that each row of tabs fills the entire width of the tab control.

@TCS_FIXEDWIDTH

All tabs are the same width. This style cannot be combined with the `@TCS_RIGHTJUSTIFY` style.

@TCS_RAGGEDRIGHT

Rows of tabs will not be stretched to fill the entire width of the control. This style is the default.

@TCS_FOCUSONBUTTONDOWN

The tab control receives the input focus when clicked.

@TCS_OWNERDRAWFIXED

The parent window is responsible for drawing tabs.

@TCS_TOOLTIPS

The tab control has a ToolTip control associated with it.

@TCS_FOCUSNEVER

The tab control does not receive the input focus when clicked.

Tab control functions and statements

[tcDeleteAllTabs](#)(window | dialog, ID)

Removes all tabs from a tab control.

[tcDeleteTab](#)(window | dialog, ID, item as INT)

Removes a tab from a tab control.

INT = [tcGetFocusTab](#)(window | dialog, ID)

Call this function to retrieve the index of the tab with the current focus.

UINT = [tcGetItemData](#)(window | dialog, ID, tab as INT)

Retrieves the application defined data associated with a tab.

INT = [tcGetRowCount](#)(window | dialog, ID)

Retrieves the current number of rows in a tab control.

INT = [tcGetSelectedTab](#)(window | dialog, ID)

Retrieves the index of the currently selected tab.

INT = [tcGetTabCount](#)(window | dialog, ID)

Retrieves the number of tabs in the control.

STRING = [tcGetTabText](#)(window | dialog, ID, item)

Returns the text of a tab.

[tcHighlightTab](#)(window | dialog, ID, item, bHighlight)

Sets the highlight state of a tab item.

[tcHighlightTab](#)(window | dialog, ID, item, bHighlight)

Sets the highlight state of a tab item.

[tcInsertTab](#)(window | dialog, ID, index, item {, iImage = -1})

Inserts a new tab in a tab control.

[tcSetFocusTab](#)(window | dialog, ID, item)

Sets the focus to a specified tab in a tab control.

[tcSetImage](#)(window | dialog, ID, tab, iImage)

Changes or removes a tabs image.

[tcSetImageList](#)(window | dialog, ID, himl)

Assigns an image list to a tab control.

[tcSetItemData](#)(window | dialog, ID, tab , value)

Sets the application-defined item data associated with a tab in the tab control.

[tcSetMinTabSize](#)(window | dialog, ID, cx)

Sets the minimum width of items in a tab control.

[tcSetSelectedTab](#)(window | dialog, ID, item)

Selects a tab in a tab control.

[tcSetTabText](#)(window | dialog, ID, item , text)

Changes the text of a tab.

[tcSetTip](#)(window | dialog, ID, tab, text)

Sets the tool tip for a tab in the tab control.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight , flags , ID

The text printed in a control will use the default font specified in the display control panel unless changed with the [SETFONT](#) statement.

[SETSIZE](#) window | dialog, L, T, W, H , ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use [@SWHIDE](#) to hide the control and [@SWRESTORE](#) to show the control

Notification messages

10.12.2!Toolbar Controls - WIP

About Toolbar controls

A *toolbar* is a control window that contains one or more buttons. Each button sends a command message to the parent window when the user chooses it. Typically, the buttons in a toolbar correspond to items in the application's menu, providing an additional and more direct way for the user to access an application's commands.

Creating the control

A toolbar control is created and added to a window with the [LOADTOOLBAR](#) function. A bitmap is used as the images for the toolbar buttons and can be loaded directly from a handle returned by [LOADIMAGE](#) or from a bitmap resource compiled in the executable. A toolbar has a control ID that allows communicating to the control with the [CONTROLCMD](#) statement. The syntax of [LOADTOOLBAR](#):

```
success = LOADTOOLBAR( window | dialog, resourceID | handle, toolbarID, buttonIDs[] ,  
numIDs, style )
```

When a button on a toolbar is pressed Windows will send an `@IDCONTROL` message to your handler subroutine with the ID of the button contained in `@CONTROLID`. The variable *buttonIDs* is an array of type INT that is dimensioned to the number of buttons in the toolbar. Each value in the array corresponds to the ID of a button or 0 for a separator. *numIDs* is the size of the *buttonIDs* array

The function returns zero on failure or non-zero for success.

The following is an example snippet of loading a toolbar with 7 button images. A separator is used between buttons 3 and 4.

Assume an open window w1

```
DEF hbitmap:UINT  
DEF tbArray[8]:INT  
tbArray = 2,3,4,0,5,6,7,8  
hbitmap = LOADIMAGE(GETSTARTPATH + "toolbar.bmp",@IMGBITMAP | @IMGMAPCOLORS)  
  
REM this toolbar is loaded from a bitmap handle  
IF LOADTOOLBAR(w1,hbitmap,998, tbArray, 8, @TBTOP | @TBFROMHANDLE)  
    CONTROLCMD w1,998,@TBENABLEBUTTON,3,0  
ENDIF
```

In the snippet above we assign an ID of 998 to the toolbar control. After the toolbar is loaded `CONTROLCMD` is used to disable a the button with an ID of 3.

Toolbar control styles

The following toolbar style flags can be specified in the [LOADTOOLBAR](#) function.

@TBTOP

Creates a toolbar that is positioned at the top of the window. This is the default

@TBBOTTOM

Creates a toolbar this is positioned at the bottom of the window

@TBRIGHT

Creates a toolbar that is positioned at the right of the window. Must be combined with @TBNORESIZE. Vertical toolbars cannot be resized automatically with @TBRESIZE. The width and height of a vertical toolbar must be set with the SETSIZE statement.

@TBLEFT

Creates a toolbar the is positioned at the left edge of the window. Must be combined with @TBNORESIZE. Vertical toolbars cannot be resized automatically with @TBRESIZE. The width and height of a vertical toolbar must be set with the SETSIZE statement.

@TBNOALIGN

Prevents the control from automatically moving to the top or bottom of the parent window. Instead, the control keeps its position within the parent window despite changes to the size of the parent window. If the @TBTOP or @TBBOTTOM style is also used, the height is adjusted to the default, but the position and width remain unchanged.

@TBWRAPABLE

Creates a toolbar control that can have multiple lines of buttons. Toolbar buttons can "wrap" to the next line when the toolbar becomes too narrow to include all buttons on the same line.

@TBFLAT

Creates a toolbar with flat buttons. The buttons and toolbar are transparent with this style. Available on systems with Internet Explorer 3.0 and above installed.

@TBLIST

Creates a toolbar with buttons that can have text to the right of the buttons. Available on systems with Internet Explorer 3.0 and above installed.

@TBNORESIZE

Prevents the control from using the default width and height when setting its initial size or a new size. Instead, the control uses the width and height specified in the request for creation or sizing. Control may then be positioned with the SETSIZE statement

@TBFROMHANDLE

Specifies the bitmap is to loaded from a handle returned by LOADIMAGE instead of from resources.

@TBTRANSPARENT

Allows non flat toolbars to have transparent backgrounds. Useable on systems with Internet Explore 4.0 or higher installed.

@TBTOOLTIPS

Allows toolbars to display a tool tip when the cursor hovers over a button. The text for the tool tips is set with the `@TBSETTIP` command.

NOTE: always check `@NOTIFYCODE` if your toolbar has tooltips. The tooltips themselves send notification messages using the same button ID. `@NOTIFYCODE` will be 0 if the button was clicked on.

Controlling the toolbar control.

The toolbar control is accessed through the `CONTROLCMD` statement/function.

`CONTROLCMD` can be used as a statement or function depending on the command issued.

Statements and Functions

[CONTROLCMD](#) window | dialog, toolbarID, `@TBRESIZE`

Use in response to the `@IDSIZE` message to automatically change the size of the toolbar to fit the window. This command is ignored if the toolbar is created with the `@TBNORESIZE` flag.

[CONTROLCMD](#) window | dialog, toolbarID, `@TBSETBITMAPSIZE`, width, height

Sets the size of the bitmap images for the buttons. The default size is 16 x 15

[CONTROLCMD](#) window | dialog, toolbarID, `@TBSETBUTTONSIZE`, width, height

Sets the size of the buttons. The default size is 24 x 23. The size of the buttons must be at least 7 pixels greater in width and height than the bitmap size otherwise the bitmap images will be clipped.

[CONTROLCMD](#) window | dialog, toolbarID, `@TBENABLEBUTTON`, buttonID, enable

Enables or disables the button specified by buttonID. use 1 to enable and 0 to disable the button. A button that is disabled can not be pressed and has a 'grayed out' appearance.

[CONTROLCMD](#) window | dialog, toolbarID, `@TBSETLABELS`, strLabels

Sets the button labels used with the `@TBLIST` and `@TBFLAT` styles. Strings consist of text labels separated by a bar '|' symbol and ending with two bar '||' symbols. Example:

```
CONTROLCMD d1, 999, @TBSETLABELS, "New|Open|Save|Cut|Copy|Paste|Print|Help||"
```

[CONTROLCMD](#) window | dialog, toolbarID, `@TBSETBUTTONSTYLE`, buttonID, style

Changes the behavior of a toolbar button. This command is only available on systems that have Internet Explorer 4.0 or greater installed. Style can be a combination of:

`@TBBUTTONCHECK` - Button stays pressed down until released by clicking on again

`@TBBUTTONGROUP` - When combined with `@TBBUTTONCHECK` creates a button that stays pressed until another button in the group is pressed.

state = [CONTROLCMD](#) (window | dialog, toolbarID, `@TBGETBUTTONSTATE`, buttonID)

Returns 1 if button is currently pressed. Only applied to buttons with the `@TBBUTTONCHECK` style.

width = [CONTROLCMD](#) (window | dialog, toolbarID, `@TBGETBUTTONWIDTH`)

Returns the width of the toolbar buttons

height = [CONTROLCMD](#) (window | dialog, toolbarID, [@TBGETBUTTONHEIGHT](#))
Returns the height of the toolbar buttons

[CONTROLCMD](#) window | dialog, toolbarID, [@TBSETTIP](#), buttonID, tiptext
Sets the tool tip text for a button. Toolbar must have been created with [@TBTOOLTIPS](#).

return = [CONTROLEXISTS](#) (window | dialog, ID)
Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1
Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)
Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)
Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)
Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)
Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)
Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)
Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID
Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg
Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text
Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID
Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight, flags, ID

The text printed in a control will use the default font specified in the display control panel unless changed with the [SETFONT](#) statement.

[SETSIZE](#) window | dialog, L, T, W, H, ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use [@SWHIDE](#) to hide the control and [@SWRESTORE](#) to show the control

Design Considerations

Horizontal toolbars are more natural to view than vertical ones. The eyes naturally follow the left to right ordering of a standard toolbar placed at the top of a window. Vertical toolbars tend to be more awkward to use.

MDI windows will automatically resize the client window to fit toolbars and status windows. A regular window places the toolbar in its client area so you need to take the height (width) of the toolbar into account when drawing to the window. Even though a vertical toolbar won't respond to the [@TBRESIZE](#) command you should still use [CONTROLCMD](#) window | dialog, ID, [@TBRESIZE](#) on vertical toolbars in an MDI window. This tells the MDI frame to reserve space for the toolbar after sizing. Use the command after resizing the toolbar with [SETSIZE](#)

Toolbars on the bottom and right will flicker slightly when resized. This is due to the toolbar needing to refresh when moved. Consider using the top and right positions.

If you need more than one toolbar at the top you will need to specify [@NORESIZE](#) and handle the resizing and positioning manually with the [SETSIZE](#) statement. [@TBRESIZE](#) only handles single toolbars on an edge.

See Also

The sample programs *loadtoolbar.iwb* and *verticaltoolbar.iwb*

10.12.2(ToolTip Controls - WIP

About ToolTip controls

Tooltips appear automatically, or pop up, when the user pauses the mouse pointer over a tool or some other UI element.

Creating the control

```
UINT = ToolTipControl(win as WINDOW, flags as INT, exStyle as INT)
```

ToolTip control styles

@TTS_ALWAYSSTIP

Indicates that the ToolTip control appears when the cursor is on a tool, even if the ToolTip control's owner window is inactive.

@TTS_NOPREFIX

Prevents the system from stripping ampersand characters from a string or terminating a string at a tab character.

@TTS_NOFADE

Disables fading ToolTip animation on Windows 2000 and above systems.

@TTS_NOANIMATE

Disables sliding ToolTip animation on Microsoft Windows 2000 and above systems.

@TTS_BALLOON

Indicates that the ToolTip control has the appearance of a cartoon "balloon".

ToolTip control functions and statements

[ttAddTool](#)(hToolTip, flags, uID, hwnd, text)

Adds a tip to a tool tip control.

[ttDeleteTool](#)(hToolTip, uID, hwnd)

Removes a tip from a tool tip control.

[ttRelayMessage](#)(hTooltip)

Passes a mouse message to a ToolTip control for processing.

[ttSetToolRect](#)(hToolTip, uid, hwnd, rc)

Sets a new bounding rectangle for a tool.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wParam, lParam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight , flags , ID

The text printed in a control will use the default font specified in the display control panel unless changed with the [SETFONT](#) statement.

[SETSIZE](#) window | dialog, L, T, W, H , ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use [@SWHIDE](#) to hide the control and [@SWRESTORE](#) to show the control

Notification messages

10.12.27TrackBar Controls - WIP

About Trackbar controls

A trackbar is a window that contains a slider (sometimes called a thumb) in a channel, and optional tick marks. When the user moves the slider, using either the mouse or the direction keys, the trackbar sends notification messages to indicate the change.

Creating the control

UINT = TrackBarControl(win as WINDOW,l as INT,t as INT,w as INT,h as INT,flags as INT,exStyle as INT,id as UINT)

Trackbar control styles

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the @GROUP style after the first control belong to the same group. The next control with the @GROUP style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSTEMMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

@TBS_AUTOTICKS

The trackbar control has a tick mark for each increment in its range of values.

@TBS_VERT

The trackbar control is oriented vertically.

@TBS_HORZ

The trackbar control is oriented horizontally. This is the default orientation.

@TBS_TOP

The trackbar control displays tick marks above the control. This style is valid only with

@TBS_HORZ.

@TBS_BOTTOM

The trackbar control displays tick marks below the control. This style is valid only with @TBS_HORZ.

@TBS_LEFT

The trackbar control displays tick marks to the left of the control. This style is valid only with @TBS_VERT.

@TBS_RIGHT

The trackbar control displays tick marks to the right of the control. This style is valid only with @TBS_VERT.

@TBS_BOTH

The trackbar control displays tick marks on both sides of the control. This will be both top and bottom when used with @TBS_HORZ or both left and right if used with @TBS_VERT.

@TBS_NOTICKS

The trackbar control does not display any tick marks.

@TBS_ENABLESELRANGE

The trackbar control displays a selection range only.

@TBS_FIXEDLENGTH

The trackbar control allows the size of the slider to be changed with the SetTrackBarThumbLength command.

@TBS_NOTHUMB

The trackbar control does not display a slider.

@TBS_TOOLTIPS

The trackbar control supports ToolTips. When a trackbar control is created using this style, it automatically creates a default Tooltip control that displays the slider's current position.

Trackbar control functions and statements

INT = [GetTrackBarLineSize](#)(window | dialog, ID)

Retrieves the number of logical positions the TrackBar's slider moves in response to keyboard input from the arrow keys.

INT = [GetTrackBarPageSize](#)(window | dialog, ID)

Retrieves the number of logical positions the trackbar's slider moves in response to keyboard input.

INT = [GetTrackBarPosition](#)(window | dialog, ID)
Retrieves the current logical position of the slider in a trackbar.

INT = [GetTrackBarRangeMax](#)(window | dialog, ID)
Returns the upper limit of the range of the trackbar control.

INT = [GetTrackBarRangeMin](#)(window | dialog, ID)
.Returns the lower limit of the range of the trackbar control.

[SetTrackBarLineSize](#)(window | dialog, ID, size)
Sets the number of logical positions the TrackBar's slider moves in response to keyboard input from the arrow keys.

[SetTrackBarPageSize](#)(window | dialog, ID, size)
Sets the page size for a trackbar control.

[SetTrackBarPosition](#)(window | dialog, ID, pos)
Sets the current logical position of the slider in a trackbar.

[SetTrackBarRange](#)(window | dialog, ID, min ,max)
Sets the range of minimum and maximum logical positions for the slider in a trackbar.

[SetTrackBarThumbLength](#)(window | dialog, ID, length)
Sets the length of the slider in a trackbar.

[SetTrackBarTickFreq](#)(window | dialog, ID, freq)
Sets the interval frequency for tick marks in a trackbar.

return = [CONTROLEXISTS](#) (window | dialog, ID)
Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1
Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)
Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)
Used to retrieve the caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)
Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wParam, lParam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLCOLOR](#) window | dialog, ID, fg, bg

Sets the text color and background color of the control.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight , flags , ID

The text printed in a control will use the default font specified in the display control panel unless changed with the [SETFONT](#) statement.

[SETSIZE](#) window | dialog, L, T, W, H , ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

Notification messages

10.12.2 Treeview Controls - WIP

About tree view controls

A *tree-view control* is a window that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk. Each item consists of a label and an optional bitmapped image, and each item can have a list of subitems associated with it.

By clicking an item, the user can expand and collapse the associated list of subitems. The following illustration shows a tree-view control that displays a table of contents.

Creating the control

Tree view controls are created in the same manner as the standard control types, either through the [Form Editor](#) or manually with the [CONTROL](#) statement using `@TREEVIEW` for the control type.

Tree view control styles

The following list view style flags can be specified in the [CONTROL](#) statement or by ticking the corresponding check box in the control properties of the [Form Editor](#):

@DISABLE

Sets the initial state of a control in a window or dialog to disabled. Controls that are disabled are grayed out and cannot be selected.

@GROUP

Specifies the first control of a group of controls in which the user can move from one control to the next with the arrow keys. All controls defined without the `@GROUP` style after the first control belong to the same group. The next control with the `@GROUP` style starts the next group (that is, one group ends where the next begins)

@TABSTOP

Moves the input focus between controls if controls are in a dialog. This style has no effect for controls created in a window.

@SYSTEMMENU

Sets the initial state of a control in a window or dialog to hidden. Hidden controls can subsequently be shown with the [SHOWWINDOW](#) command.

```
if getstate( t,1039 ) then style+="@BORDER!"
```

@TVSDISABLEDRAGDROP

Prevents the tree-view control from sending `@TVNBEGINDRAG` notification messages.

@TVSEDLABELS

Allows the user to edit the labels of tree-view items.

@TVSFULLROWSELECT

Enables full-row selection in the tree view. The entire row of the selected item is highlighted, and clicking anywhere on an item's row causes it to be selected. This style cannot be used in conjunction with the `@TVSHASLINES` style.

@TVSHASBUTTONS

Displays plus (+) and minus (-) buttons next to parent items. The user clicks the buttons to expand or collapse a parent item's list of child items. To include buttons with items at the root of the tree

view, `@TVSLINESATROOT` must also be specified.

@TVSHASLINES

Uses lines to show the hierarchy of items.

@TVSINFOTIP

Obtains ToolTip information by sending the `@TVNGETINFOTIP` notification.

@TVSLINESATROOT

Uses lines to link items at the root of the tree-view control. This value is ignored if `@TVSHASLINES` is not also specified.

@TVSNOHSCROLL

Disables horizontal scrolling in the control. The control will not display any horizontal scroll bars.

@TVSNONEVENHEIGHT

Sets the height of the items to an odd height with the `TVM_SETITEMHEIGHT` message. By default, the height of items must be an even value.

@TVSNOSCROLL

Disables both horizontal and vertical scrolling in the control. The control will not display any scroll bars.

@TVSNOTOOLTIPS

Disables tool tips.

@TVSRTLREADING

Causes text to be displayed from right-to-left (RTL). Usually, windows display text left-to-right (LTR). Windows can be mirrored to display languages such as Hebrew or Arabic that read RTL. Typically, tree-view text is displayed in the same direction as the text in its parent window. If `@TVSRTLREADING` is set, tree-view text reads in the opposite direction from the text in the parent window.

@TVSSHOWSELALWAYS

Causes a selected item to remain selected when the tree-view control loses focus.

@TVSSINGLEEXPAND

Causes the item being selected to expand and the item being unselected to collapse upon selection in the tree view. If the user holds down the CTRL key while selecting an item, the item being unselected will not be collapsed.

@TVSTRACKSELECT

Enables hot tracking in a tree view control.

Statements and functions for controlling the tree view

handle = [tvInsertItem](#)(window | dialog, ID, strText, parent)

Inserts a new item into a tree view control. If *parent* is NULL then the item is inserted at the ROOT level.

Returns a handle to the new item.

handle = [tvGetSelectedItem](#)(window | dialog, ID)

Returns a handle to the currently selected item if any

success = [tvGetItemText](#)(window | dialog, ID, handle, pstrText, cchTextMax)

Retrieves the text of an item. pstrText is the string to receive the text and cchTextMax is the dimension of the string.

Returns TRUE if text could be retrieved or FALSE if your string is too small.

success = [tvSetItemText](#)(window | dialog, ID, handle, strText)

Sets the text of an item

Returns TRUE on success, FALSE if item doesn't exist.

success = [tvDeleteItem](#)(window | dialog, ID, handle)

Deletes an item in the tree view

Returns TRUE on success, FALSE if item doesn't exist.

success = [tvDeleteAllItems](#)(window | dialog, ID)

Removes all items in the tree view

Returns TRUE on success, FALSE otherwise

success = [tvSelectItem](#)(window | dialog, ID, handle)

Selects an item in the tree view

Returns TRUE on success, FALSE otherwise

data = [tvGetItemData](#)(window | dialog, ID, handle)

Retrieves a user defined unsigned integer from an item.

[tvSetItemData](#)(window | dialog, ID, handle, nData)

Stores a user defined unsigned integer in an item.

return = [CONTROLEXISTS](#) (window | dialog, ID)

Returns 1 if the control with ID exists in the window or dialog.

[ENABLECONTROL](#) window | dialog, ID, 0 | 1

Disables or enables the control. If the control is disabled it will not receive input focus when clicked on.

handle = [GETCONTROLHANDLE](#)(window | dialog, ID)

Returns the windows handle (HWND) of a control in a dialog or window.

text\$ = [GETCONTROLTEXT](#) (window | dialog, ID)

Used to retrieve the hidden caption of a control.

[GETSIZE](#)(window | dialog, L, T, W, H,, ID)

Gets the size of a control.

[MODIFYEXSTYLE](#)(window | dialog, add , remove, ID)

Adds or removes extended styles from a control.

[MODIFYSTYLE](#)(window | dialog, add, remove, ID)

Adds or removes styles from a control.

[REDRAWFRAME](#)(window | dialog, ID)

Redraws the control after the style has been modified.

[SENDMESSAGE](#) window | dialog, msg, wparam, lparam , ID

Sends a message to the control for advanced functionality.

[SETCONTROLTEXT](#) window | dialog, ID, text

Used to change the hidden caption of a control.

[SETFOCUS](#) window | dialog, ID

Gives the control the input focus.

[SETFONT](#) window | dialog, typeface, height, weight , flags , ID

Changes the font of all text in the control.

[SETSIZE](#) window | dialog, L, T, W, H , ID

Changes the size of the control.

[SHOWWINDOW](#) window | dialog, flags, ID

Changes the visibility state of the control. Use `@SWHIDE` to hide the control and `@SWRESTORE` to show the control

```
CONST TV_FIRST = 0x1100
```

```
CONST TVM_SETBKCOLOR = (TV_FIRST + 29)
```

```
CONST TVM_SETTEXTCOLOR = (TV_FIRST + 30)
```

```
sendmessage win,TVM_SETTEXTCOLOR,0,rgb(255,0,0),win_TREEVIEW1
```

```
sendmessage win,TVM_SETBKCOLOR,0,rgb(0,128,128),win_TREEVIEW1
```

Notification messages

A tree view control sends notification messages to the parent window or dialog in the `@NOTIFYCODE` variable. The ID of the control is found in `@CONTROLID` the same as standard controls. The following notification messages are supported:

`@TVNBEGINDRAG`

Notifies a tree-view control's parent window that a drag-and-drop operation involving the left mouse button is being initiated.

@TVNBEGINLABELEDIT

Notifies a tree-view control's parent window about the start of label editing for an item.

@TVNBEGINRDRAG

Notifies a tree-view control's parent window about the initiation of a drag-and-drop operation involving the right mouse button.

@TVNDELETEITEM

Notifies a tree-view control's parent window that an item is being deleted.

@TVNENDLABELEDIT

Notifies a tree-view control's parent window about the end of label editing for an item.

@TVNGETDISPINFO

Requests that a tree-view control's parent window provide information needed to display or sort an item.

@TVNGETINFOTIP

Sent by a tree-view control that has the @TVSINFOTIP style. This notification is sent when the control is requesting additional text information to be displayed in a ToolTip.

@TVNITEMEXPANDED

Notifies a tree-view control's parent window that a parent item's list of child items has expanded or collapsed.

@TVNITEMEXPANDING

Notifies a tree-view control's parent window that a parent item's list of child items is about to expand or collapse.

@TVNKEYDOWN

Notifies a tree-view control's parent window that the user pressed a key and the tree-view control has the input focus.

@TVNSELCHANGED

Notifies a tree-view control's parent window that the selection has changed from one item to another.

@TVNSELCHANGING

Notifies a tree-view control's parent window that the selection is about to change from one item to another.

@TVNSETDISPINFO

Notifies a tree-view control's parent window that it must update the information it maintains about an item.

@TVNSINGLEEXPAND

Sent by a tree-view control with the @TVSSINGLEEXPAND style when the user opens or closes a tree item using a single click of the mouse.

UDT's used with the tree view control.

Certain notification messages send a pointer to a UDT in @LPARAM. Access the UDT using C style pointer dereferencing. The NMHDR type is a built in UDT and is common to all notification messages:

```
TYPE NMHDR
    DEF hwndFrom as UINT
    DEF idFrom as INT
    DEF code as INT
ENDTYPE
```

The NMTVGETINFOTIP UDT contains and receives tree-view item information needed to display a ToolTip for an item. This structure is used with the @TVNGETINFOTIP notification message.

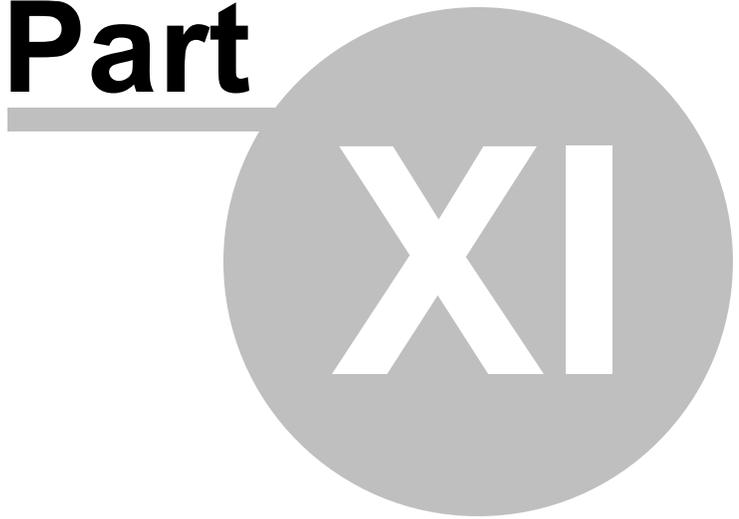
```
TYPE NMTVGETINFOTIP
    NMHDR hdr
    POINTER pszText
    int cchTextMax
    UINT hItem
    UINT lParam
ENDTYPE
```

The NMTVKEYDOWN UDT contains information about a keyboard event in a tree-view control. This structure is used with the @TVNKEYDOWN notification message.

```
TYPE NMTVKEYDOWN {
    NMHDR hdr
    WORD wVKey
    UINT flags
}
ENDTYPE
```


How-To

Part

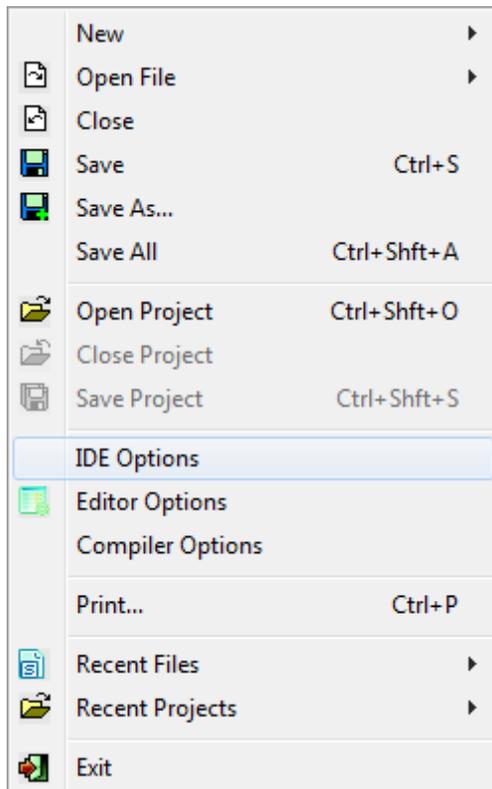


11 How-To

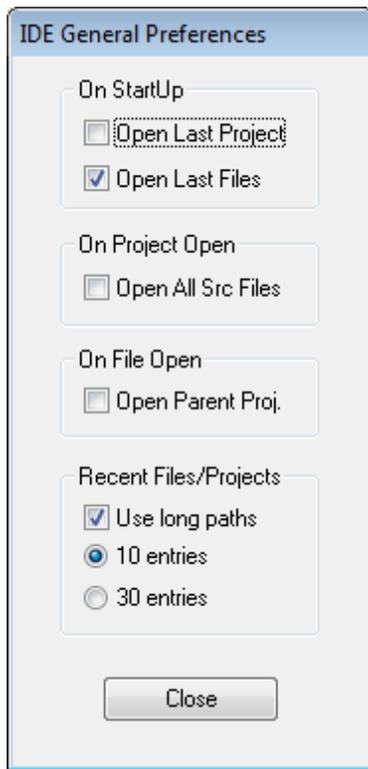
This section, and all its sub-sections, describes how to perform various activities.

11.1 Set Startup Preferences

There are several general IDE options available to the User.



Selecting *File / IDE Options* from the [Main Menu](#) opens the *IDE General Preferences* dialog, shown below.



The *IDE General Preferences* dialog has four sections of options.

Each option is discussed below.

On Startup

- *Open Last Project* - when checked, will cause the last opened project to be automatically opened each time the IDE is opened. This action is regardless of whether or not the project was actually open when the IDE was closed.
- *Open Last Files* - when checked, will cause all files that were open when the IDE was last closed to be automatically opened when the IDE is opened.

On Project Open

- *Open All Src Files* - when checked, will cause all source files belonging to a project to automatically be opened when the project is opened.

On File Open

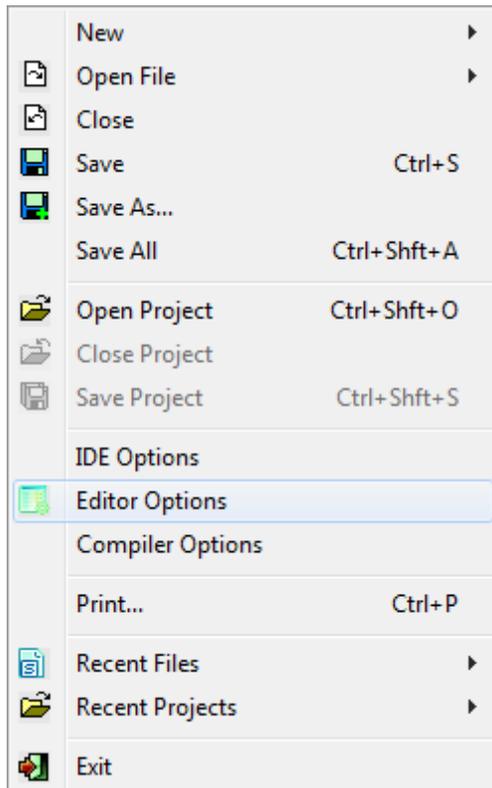
- *Open Parent Proj* - when checked, will cause the parent project, if applicable, to be automatically be opened. If a project is opened via this method the project's source files will not be opened even if the previous option has been checked.

Recent Files/Projects

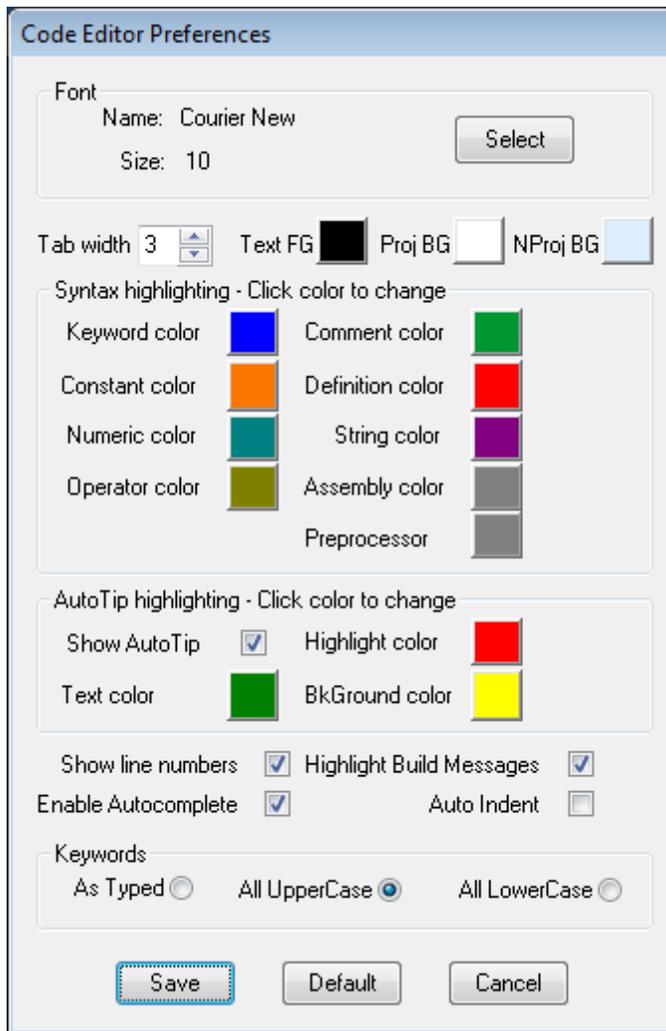
- *Use long paths* - when checked, will cause menu entries to be complete path names. When unchecked, portions of the path name may be replaced by '...'.
 - *10 entries* - when selected, the *Files* and *Projects* sub-menus will be limited to the last 10 opened files. This is the default.
 - *30 entries* - when selected, the *Files* and *Projects* sub-menus will be limited to the last 30 opened files.

11.2 Set Editor Preferences

Many of the default characteristics of the [Code Editor](#) windows can be set by the User.



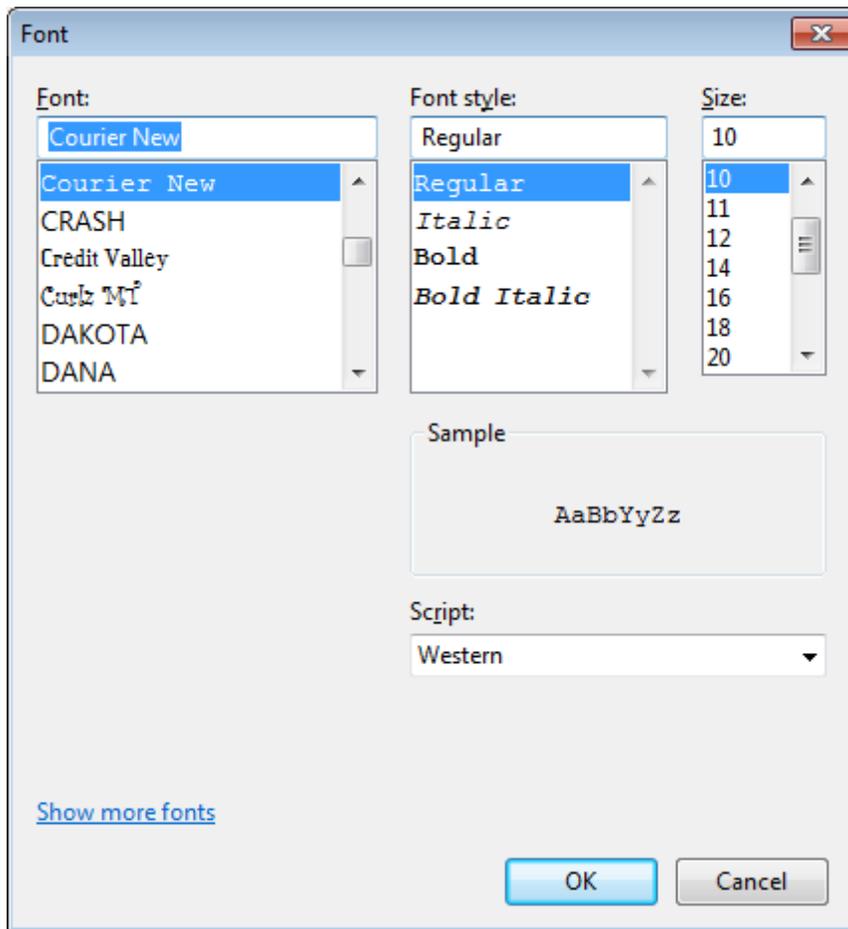
Selecting *File / Editor Options* from the [Main Menu](#) opens the *Code Editor Preferences* dialog, shown below.



The *Code Editor Preferences* dialog has 20+ options.

Each option is discussed in the table below.

Option	Description
Font Name	The name of the font currently be used to display all text in the Code Editor and Output windows. A mono-spaced, or fixed font is recommended for programming use. Some True Type fonts do not display line numbers well. Click the <i>Select</i> button to change.
Font Size	The size of the font currently be used to display all text in the Code Editor and Output windows. Click the <i>Select</i> button to change.
Select	Opens the Font dialog, shown below, which allows the User to change the font name and size.



The User selects the desired font and size to be used in all [Code Editor](#) and [Output](#) windows.

Clicking *OK* saves any changes
Clicking *Cancel* discards any changes.

Option	Description
Tab Width	Sets the width of the <i>TAB</i> key in character spaces.
Text FG	The default color for text in the Code Editor window. The color button indicates the currently selected color. Clicking the button opens the standard <i>Color</i> dialog, shown below, where the User may select a new color.
Proj BG	The default color for the background in the Code Editor window. The color button indicates the currently selected color. Clicking the button opens the standard <i>Color</i> dialog, shown below, where the User may select a new color.
NProjBG	Not Used - For Future Use

Option	Description
--------	-------------

Keyword Color	The default color for language keywords (commands, functions) in the Code Editor window. The color button indicates the currently selected color. Clicking the button opens the standard <i>Color</i> dialog, shown below, where the User may select a new color.
Comment Color	The default color for commented text (via a ' or /* */ block) in the Code Editor window. The color button indicates the currently selected color. Clicking the button opens the standard <i>Color</i> dialog, shown below, where the User may select a new color.
Constant Color	The default color for a defined constant in the Code Editor window. The color button indicates the currently selected color. Clicking the button opens the standard <i>Color</i> dialog, shown below, where the User may select a new color.
Definition Color	The default color for indicating variable types in the Code Editor window. The color button indicates the currently selected color. Clicking the button opens the standard <i>Color</i> dialog, shown below, where the User may select a new color.
Numeric Color	The default color for literal numbers in the Code Editor window. The color button indicates the currently selected color. Clicking the button opens the standard <i>Color</i> dialog, shown below, where the User may select a new color.
String Color	The default color for quoted text strings in the Code Editor window. The color button indicates the currently selected color. Clicking the button opens the standard <i>Color</i> dialog, shown below, where the User may select a new color.
Operator Color	The default color for the various language operators in the Code Editor window. The color button indicates the currently selected color. Clicking the button opens the standard <i>Color</i> dialog, shown below, where the User may select a new color.
Assembly Color	The default color for assembly code text in the Code Editor window. The color button indicates the currently selected color. Clicking the button opens the standard <i>Color</i> dialog, shown below, where the User may select a new color.
Preprocessor	The default color for language preprocessor text in the Code Editor window. The color button indicates the currently selected color. Clicking the button opens the standard <i>Color</i> dialog, shown below, where the User may select a new color.

Option	Description
Show AutoTip	When checked, the <i>AutoTip</i> feature is activated. See Utilities»Code Editor»AutoTip for details of operation.

Highlight Color	The default color for highlighted text in the <i>AutoTip</i> of the Code Editor window. The color button indicates the currently selected color. Clicking the button opens the standard <i>Color</i> dialog, shown below, where the User may select a new color. See Utilities»Code Editor»AutoTip for details of operation.
Text Color	The default color for text in the <i>AutoTip</i> of the Code Editor window. The color button indicates the currently selected color. Clicking the button opens the standard <i>Color</i> dialog, shown below, where the User may select a new color. See Utilities»Code Editor»AutoTip for details of operation.
BkGround Color	The default color for the background in the <i>AutoTip</i> of the Code Editor window. The color button indicates the currently selected color. Clicking the button opens the standard <i>Color</i> dialog, shown below, where the User may select a new color. See Utilities»Code Editor»AutoTip for details of operation.

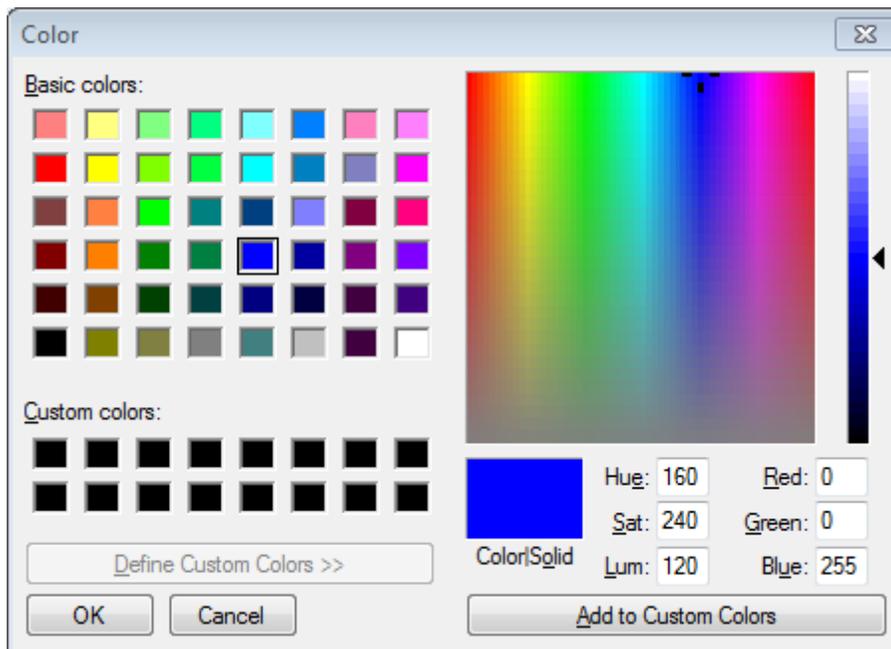
Option	Description
Show line numbers	When checked line numbers will be displayed on the left edge of the Code Editor window. The Bookmark feature will work regardless of whether line numbers are displayed or not.
Highlight Build Messages	When checked, build messages in the Output Window will be highlighted.
Enable Autocomplete	When checked the <i>AutoComplete</i> feature is activated. See Utilities»Code Editor»AutoComplete for details of operation.
Auto Indent	When checked the <i>Auto Indent</i> feature is activated. When active, pressing the <i>ENTER</i> key will cause the caret to move to the next line and to the same column as the indent level of the previous line. Otherwise the caret would move to the first column.

Option	Description
As Typed	When selected, the case of typed text will be displayed properly following the state of the SHIFT and CAPSLOCK keys.
AllUpperCase	When selected, the case of typed text will be displayed in upper case regardless of the state of the SHIFT and CAPSLOCK keys.

AllLowerCase	When selected, the case of typed text will be displayed in lower case regardless of the state of the SHIFT and CAPSLOCK keys.
---------------------	---

Note: Text is always stored exactly as it was typed; following the state of the SHIFT and CAPSLOCK keys.

Option	Description
Save	Clicking this button will save all pending changes and close the dialog.
Default	Clicking this button will reset all settings/options to the 'factory' default.
Cancel	Clicking this button will discard all pending changes and close the dialog.

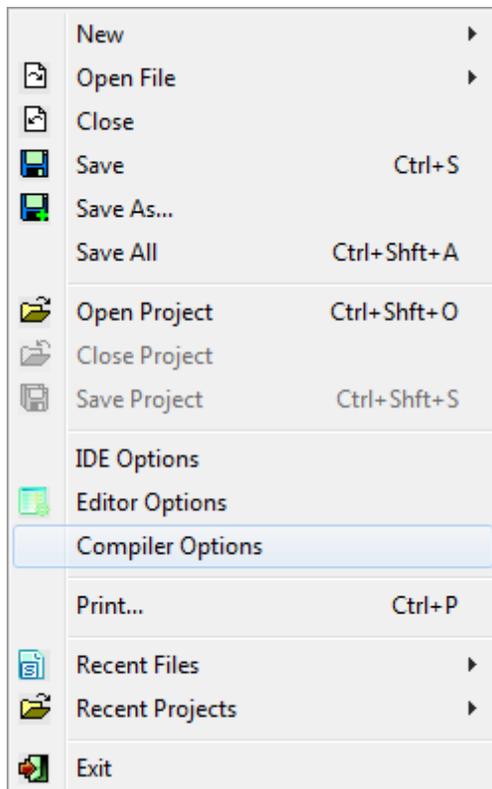


The Color dialog is used to change each of the colors mentioned above.

11.3 Set Compiler Preferences

IWBasic provides three methods for setting compiler option. Two are discussed in the [Language/Compiler Options](#) sections. This section describes the third method. Options selected here are considered global options because the setting are stored in the IWBasic.ini file and remain set between sessions. The other two methods are temporary over-rides for these global settings and do not affect the ini file.

For backward compatibility the default setting for all options, except [Command Paks](#), in this section is OFF or blank



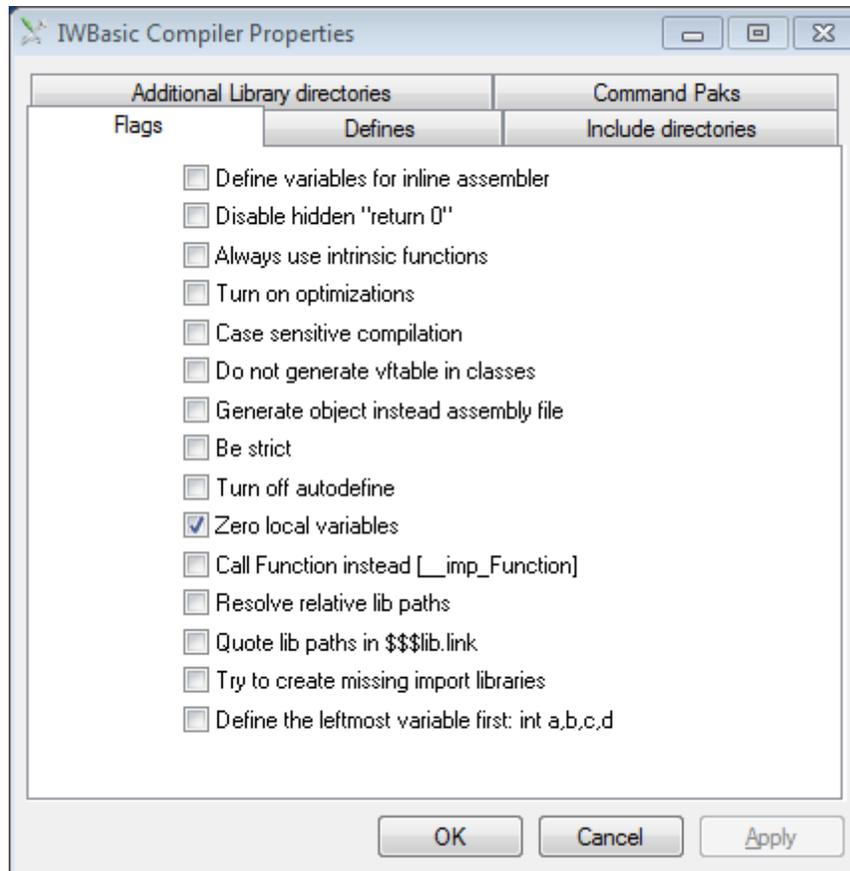
Selecting *File / Compiler Options* from the [Main Menu](#) opens the *Compiler Properties* dialog, shown below. This dialog contains five pages:

- [Flags](#)
- [Defines](#)
- [Include directories](#)
- [Additional Library directories](#)
- [Command Paks](#)

each of which is described in the following sub-sections.

The User can move between the pages by clicking the desired tab at the top of the dialog.

Flags



The *Flags* page has 15 options that can be turned off/on.

Each is listed in the table below.

The effect of each of these option flags is described in the [\\$Options keyword](#) section under the indicated name (or as a note in this section).

Flag	\$Option Keyword	Command Line
Define variables for inline assembler	AsmVariables - /p	/P
Disable hidden "return 0"	Return	
Always use intrinsic functions	Intrinsic	
Turn on optimizations	Optimization	
Case sensitive compilation	CaseSensitive	
Do not generate vtable in classes	NoVtable	

Generate object instead of assembly file	/a	/A
Be strict	Strict	/S
Turn off autodefine	<i>Note 1</i>	
Zero local variables	ZeroVariables	
Call Function instead [_imp_Function]	<i>Note 2</i>	
Resolve relative lib paths	ResolveLibs	/G
Quote lib paths in \$\$\$lib.link	QuoteLibPaths	
Try to create missing import libraries	<i>Note 3</i>	
Define leftmost variable first	DimOrder	

Note 1: Performs same function as the [AUTODEFINE](#) command.

Note 2: By default, when calling imported function, the compiler generates

```
call [__imp_FunctionName]
```

This option changes the code to

```
call FunctionName
```

Both cases are handled by the linker

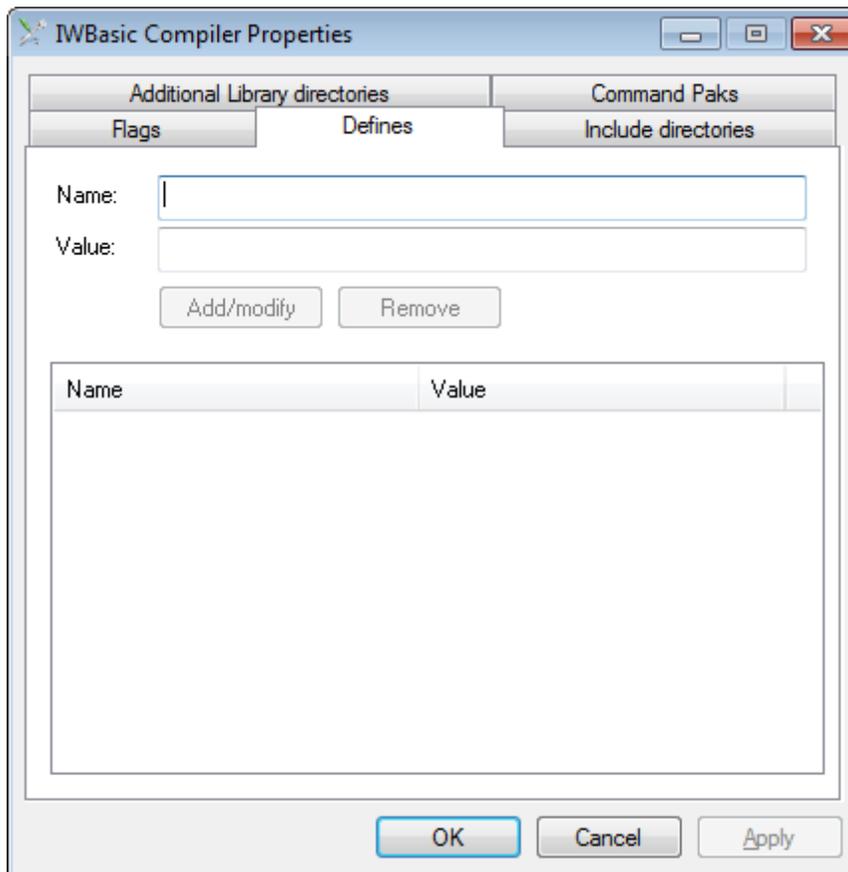
Note 3: Will execute the *libtool.exe* tool (same as selecting the *Tools/Create Import Library* option from the [Main Menu](#)) if the parser fails to find a library passed via \$use. The library will be created in the /libs directory.

At any time the User may save any pending changes by clicking the *OK* or *Apply* buttons. Both will save the changes to the *IWBasic.ini* file. *OK* closes the dialog while *Apply* leaves it open so that additional editing may be done.

Note: *OK/Apply* saves all pending changes and not just those on the currently selected page of the dialog.

Clicking *Cancel* will cancel any unsaved changes and close the dialog.

Defines



The *Defines* page allows the user to define global program constants (with or without an associated value). Constants created using this page are project global. The functionality this page provides is the same as the /D command line switch. See the [Command line](#) section for additional information.

Constants created with a value can be tested with IF, SELECT, and similar commands at runtime. Attempting to use IF, SELECT, and similar commands with a constant that hasn't been assigned a value will cause a compiler error.

As with all constants the value, if assigned, has to be an integer.

Constants, with or without an assigned value, can be tested with \$IFDEF and similar pre-directives at compile time.

Adding a Constant

- Enter a name that is unique across all programs that will be compiled using this constant.
- Enter an optional integer value
- Click the *Add/Modify* button.

The new entry will be added to the listbox. The entry will be preceded by a checkbox which controls whether or not the constant will be used in subsequent compiles.

To add additional constants the User clicks in the listbox below the last entry (to de-select all entries) then repeats the above steps.

Editing a Constant

- Click on the desired entry in the listbox
- Change the name and/or value, as desired
- Click the *Add/Modify* button.

Deleting a Constant

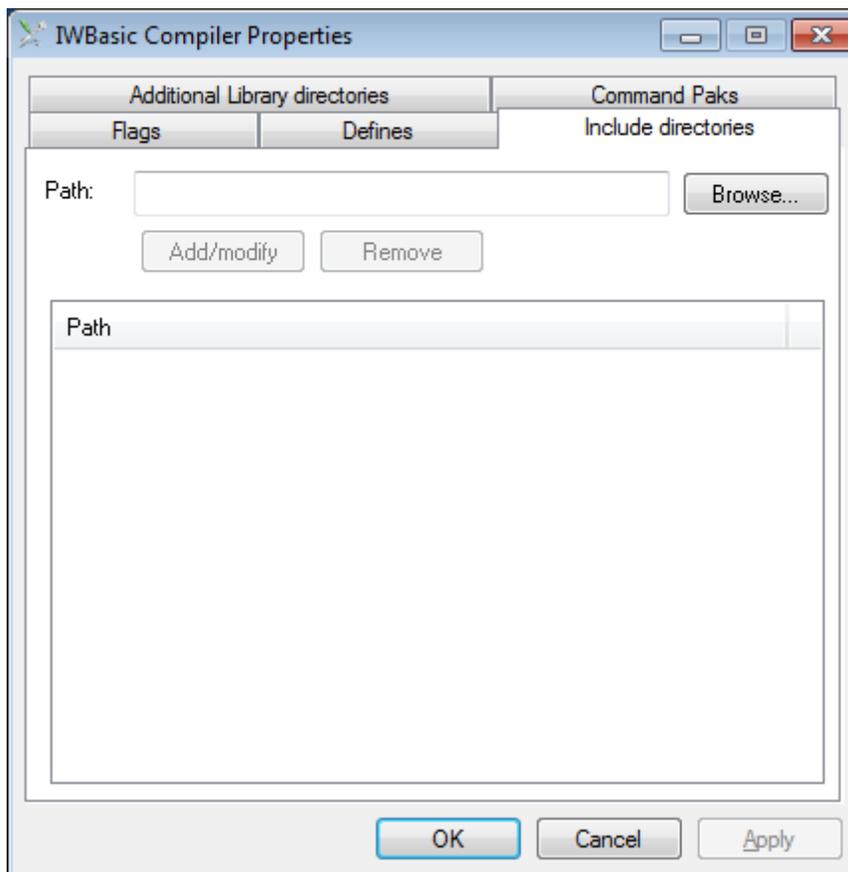
- Click on the desired entry in the listbox
- Click the *Remove* button

At any time the User may save any pending changes by clicking the *OK* or *Apply* buttons. Both will save the changes to the *IWBasic.ini* file. *OK* closes the dialog while *Apply* leaves it open so that additional editing may be done.

Note: *OK/Apply* saves all pending changes and not just those on the currently selected page of the dialog.

Clicking *Cancel* will cancel any unsaved changes and close the dialog.

Include directories



The *Include directories* page allows the user to define additional search paths for include files. The functionality this page provides is the same as the `/I` command line switch. See the [Command line](#) section for additional information.

Include files (files added to a source file via `$INCLUDE`) may or may not have full paths entered in the source file.

For those include files that don't have a full path name the search order is:

1. The directory of currently parsed file
2. The IWBDDev\Include directory

3. User directory list added in the [/i command lines switch](#) , "[IncludePath](#)" option, or this page.
4. The IWBDDev\Bin directory
5. The list from INCLUDE environment variable.

The format is same as in PATH variable: "c:\dir1;d:\dir2;e:\dir2\dir 5; ... z\last dir"

Adding an Include directory

- Enter a full path name to a folder or click the *Browse* button and navigate to the desired folder.
- Click the *Add/Modify* button.

The new entry will be added to the listbox.

To add additional paths the User clicks in the listbox below the last entry (to de-select all entries) then repeats the above steps.

Editing an Include directory

- Click on the desired entry in the listbox
- Change the path directly or click the *Browse* button and navigate to the desired folder.
- Click the *Add/Modify* button.

Deleting an Include directory

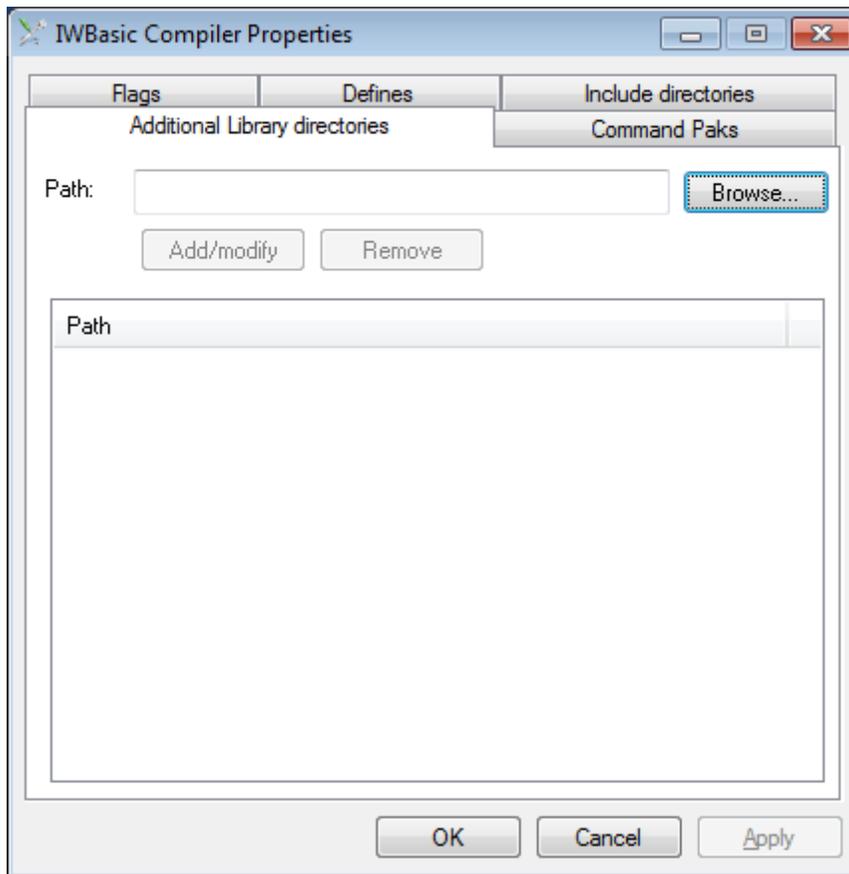
- Click on the desired entry in the listbox
- Click the *Remove* button

At any time the User may save any pending changes by clicking the *OK* or *Apply* buttons. Both will save the changes to the *IWBasic.ini* file. *OK* closes the dialog while *Apply* leaves it open so that additional editing may be done.

Note: *OK/Apply* saves all pending changes and not just those on the currently selected page of the dialog.

Clicking *Cancel* will cancel any unsaved changes and close the dialog.

Additional Library directories



The *Additional Library directories* page allows the User to define additional search paths for library files. The functionality this page provides is the same as the *LibPath* option. See the [\\$Option keyword](#) section for additional information.

Library files used in a source file (defined via \$USE) may or may not have full paths entered in the source file.

For those include files that don't have a full path name the search order is:

1. The directory of currently parsed file
2. The IWBDDev\Libs directory
3. User directory list added in the "[LibPath](#)" option or this page.

Adding a Library directory

- Enter a full path name to a folder or click the *Browse* button and navigate to the desired folder.
- Click the *Add/Modify* button.

The new entry will be added to the listbox.

To add additional paths the User clicks in the listbox below the last entry (to de-select all entries) then repeats the above steps.

Editing a Library directory

- Click on the desired entry in the listbox
- Change the path directly or click the *Browse* button and navigate to the desired folder.
- Click the *Add/Modify* button.

Deleting a Library directory

- Click on the desired entry in the listbox

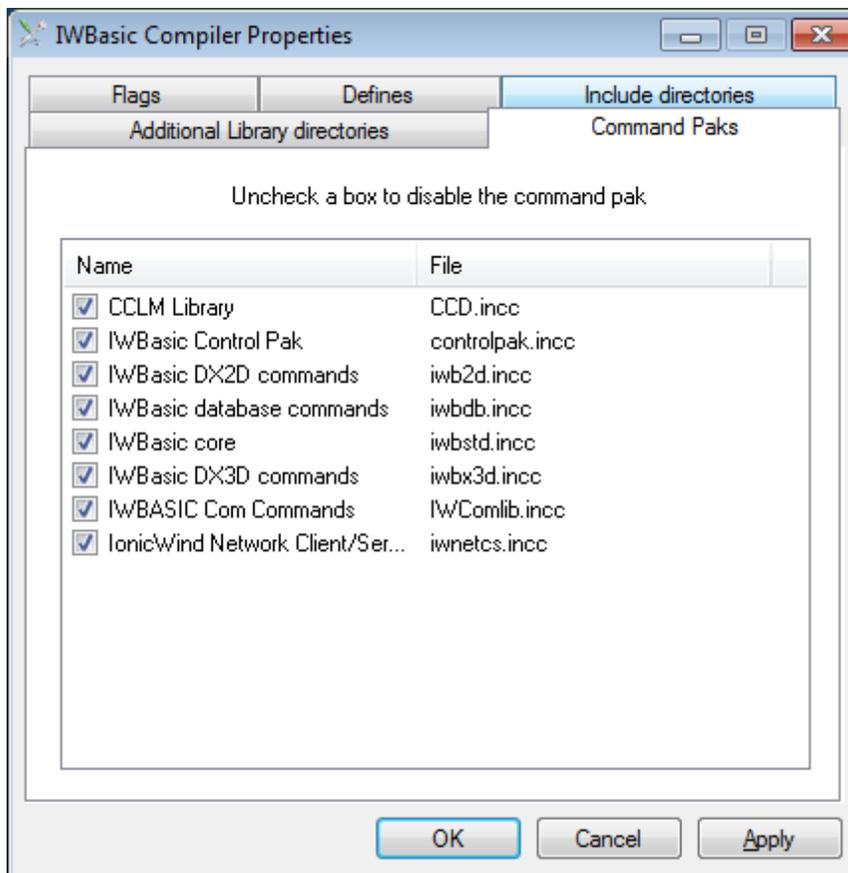
- Click the *Remove* button

At any time the User may save any pending changes by clicking the *OK* or *Apply* buttons. Both will save the changes to the *IWBasic.ini* file. *OK* closes the dialog while *Apply* leaves it open so that additional editing may be done.

Note: *OK/Apply* saves all pending changes and not just those on the currently selected page of the dialog.

Clicking *Cancel* will cancel any unsaved changes and close the dialog.

Command Paks



The *Command Paks* page lists all the add-on Command Paks that have been installed with the current installation of IWBasic.

Each pak can be enabled/disabled via its associated checkbox. This functionality is provided to enabled the User to remove an unneeded pak from the current compile process that is in a naming conflict with a function in a third-party library.

Note: Disabling a pak removes all commands contained in the pak.

At any time the User may save any pending changes by clicking the *OK* or *Apply* buttons. Both will save the changes to the *IWBasic.ini* file. *OK* closes the dialog while *Apply* leaves it open so that additional editing may be done.

Note: *OK/Apply* saves all pending changes and not just those on the currently selected page of the dialog.

Clicking *Cancel* will cancel any unsaved changes and close the dialog.

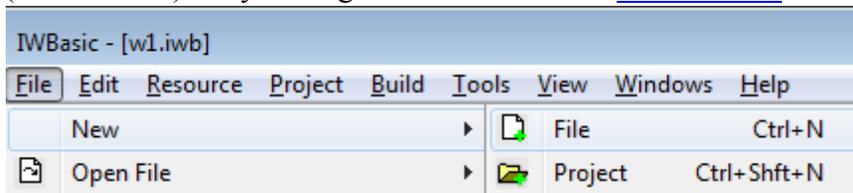
11.4 Files

This section describes how to perform the following actions on files.

- [Create a File](#)
- [Open a File](#)
- [Save a File](#)
- [Close a File](#)
- [Find Text in a File](#)
- [Replace Text in a File](#)

11.4.1 Create a File

The User can create a new, blank source file by selecting *File / New / File* from the [Main Menu](#) (shown below) or by clicking the  button on the [Main Toolbar](#).



Either method will open a new instance of the [Code Editor](#) window in the [Workspace](#).

A temporary file name will be automatically generated by the application. This file name consists of a base name and a sequence number with the default *.iwb* file extension. The sequence starts over each time the IDE is restarted. The filename has no path information associated with it.

At this point the file only exists within the [Code Editor](#) window and no actual file exists on a disk.

The file is ready for editing.

If the User initiates any activity or command that causes the save file function to be activated the User will be prompted to change the file name and provide a path to where the new file should be saved. See [How-To»Files»Save a File](#) for more information.

Note: The User always has the option of using any third-party text editor to create a text file in the desired location with the proper filename and then open the resulting file in the IDE.

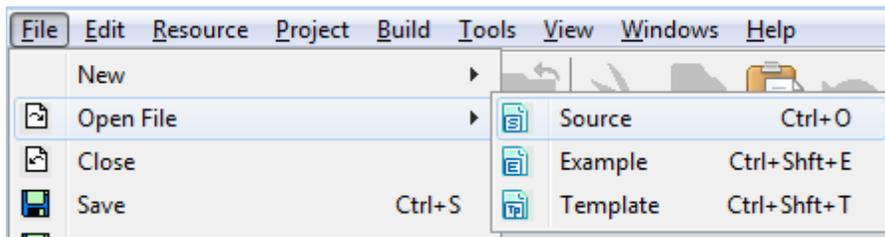
11.4.2 Open a File

There are several direct and indirect ways the User can open a source file with IWBasic.

The direct ways include;

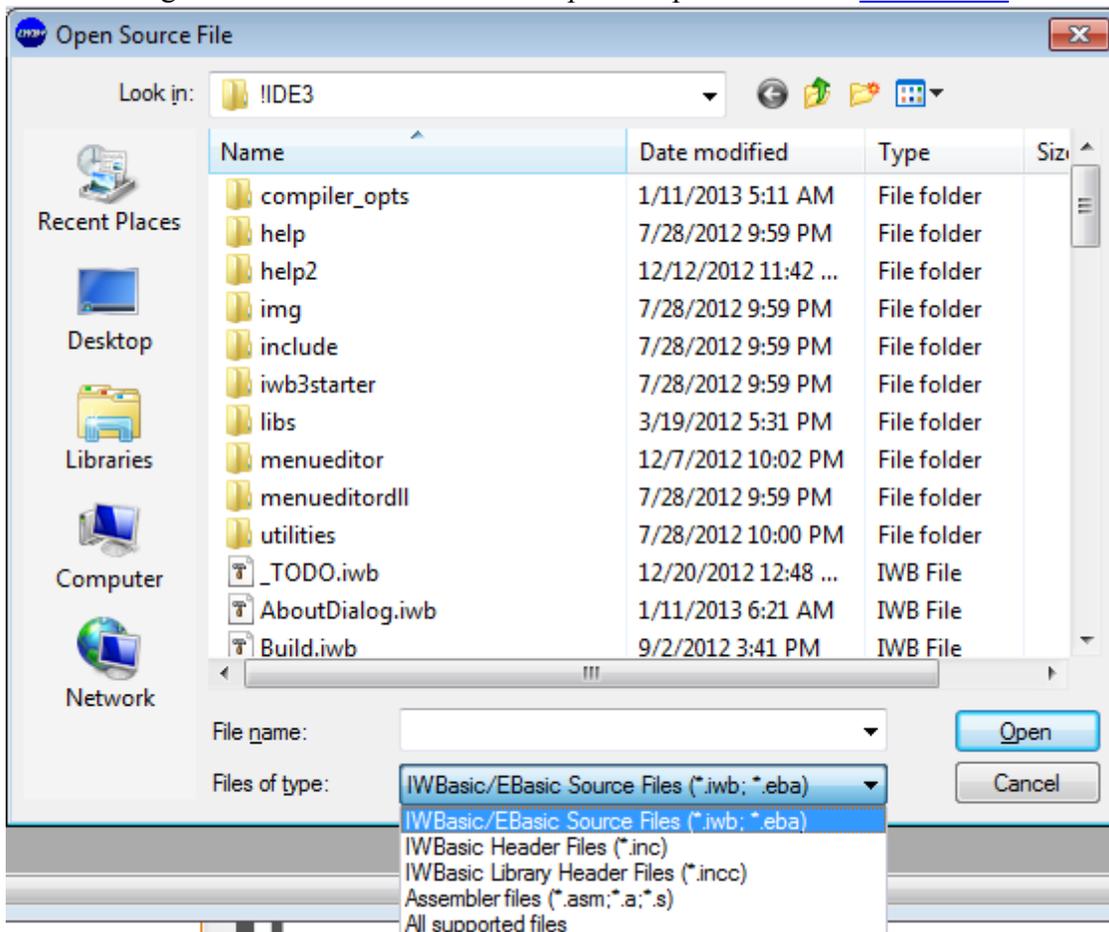
- Select *File / Open File / Source* from the [Main Menu](#) (shown below).
- Select *File / Open File / Example* from the [Main Menu](#) (shown below).
- Select *File / Open File / Template* from the [Main Menu](#) (shown below).

- Click the  button on the [Main Toolbar](#).



Each of the above actions will result in opening the *Open Source File* dialog shown below. The User may change the displayed available files by changing the selection in the *Files of Type* combo box.

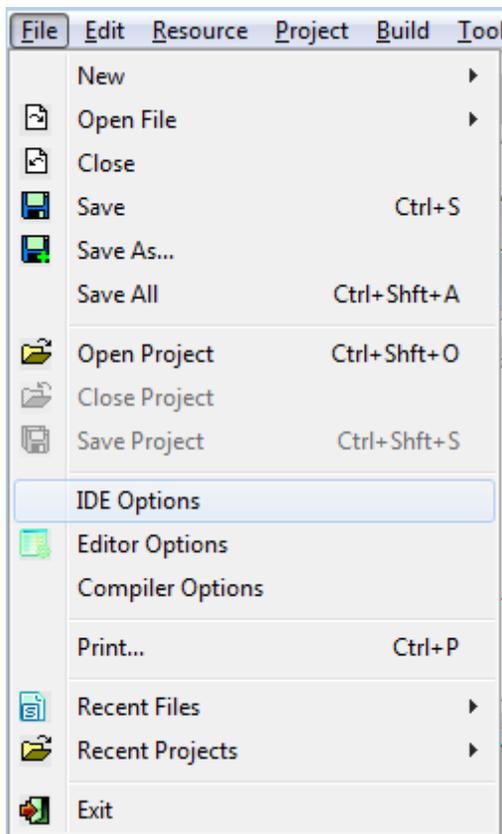
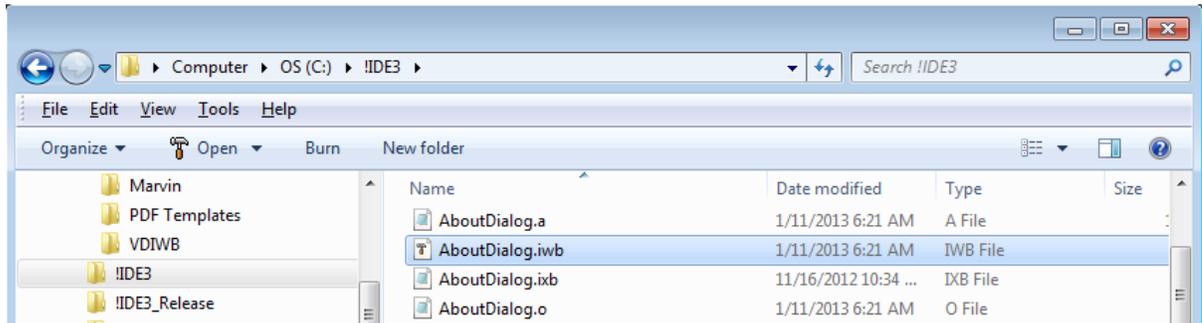
After selecting the desired file the User clicks *Open* to open the file in a [Code Editor](#) window.



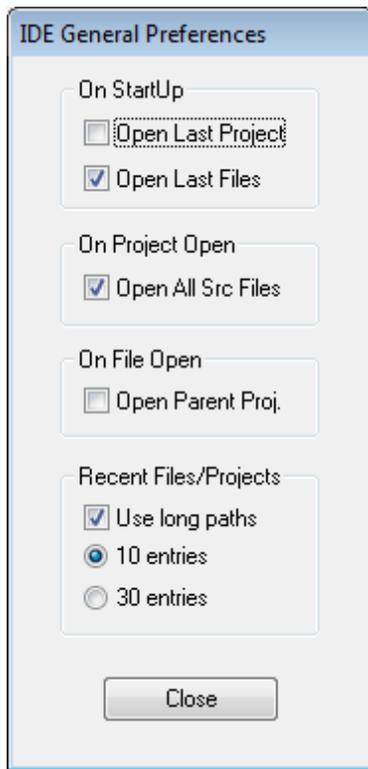
Another way to open a source file directly is to open *Windows Explorer* (shown below), navigate to the desired file, and then double-click on the file name. This action will cause the IDE to open, if not already open, and the selected file to be opened in a [Code Editor](#) window.

Note: This method will only work if the User has registered the file extension via the [Register File](#)

[Extensions](#) utility.



The User can open files indirectly by setting options in the *IDE General Preferences* dialog (shown below) which is accessed via the *File / IDE Options* option on the [Main Menu](#).



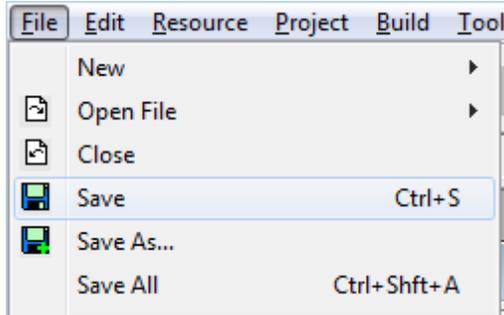
The *On Startup / Open Last Files* option, when selected, will reopen all source files, when the IDE is opened, that were open the last time the IDE was closed.

The *On Project Open / Open All Src Files* option, when selected, will open all project associated source files when the project itself is opened.

See the [How-To»Set Startup Preferences](#) section for additional information.

11.4.3 Save a File

There are several ways the User can save a source file with IWBASIC.



The [Main Menu](#) provides three ways to save a file:

- [Save](#)
- [Save As...](#)
- [Save All](#)

Each is described below.

Save

Clicking the *Save* option saves any pending changes in the currently selected file being displayed in the [Workspace](#). If the file is a new file with an automatically supplied name that has never been save the User will be presented first with the *Save As* dialog. The dialog will allow the User to rename the file (or not) and to save the file (or not).

Save As...

Clicking the *Save As* option opens the *Save As* dialog, regardless of whether or not there are any pending changes. If the User selects a new file name and does not cancel the save, any pending changes in the currently selected [Code Editor](#) window will be saved to the new file only and the new file [Code Editor](#) window will replace the previous file's [Code Editor](#) window in the [Workspace](#).

Save All

licking the *Save All* option saves any pending changes in all currently opened files in the [Workspace](#). If any file is a new file with an automatically supplied name that has never been save the User will be presented first with the *Save As* dialog. The dialog will allow the User to rename the file (or not) and to save the file (or not).

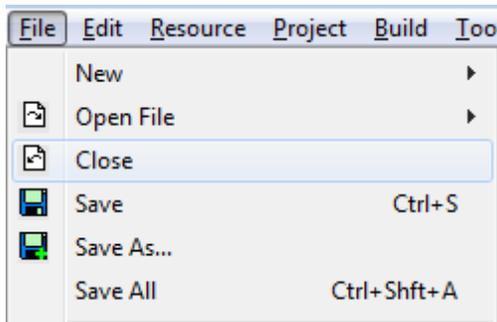
The [Main Toolbar](#) provides two ways of saving files:

- Clicking the  button performs the same action as the *Save* menu option described above.
- Clicking the  button performs the same action as the *Save All* menu option described above.

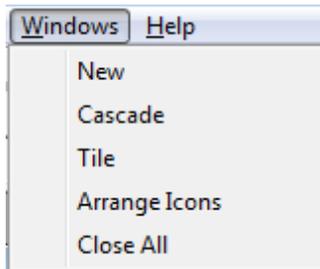
11.4.4 Close a File

There are several ways the User can close a source file with IWBASIC.

The [Main Menu](#) provides two ways to close a file:



The User can click the *Close* option on the *File* menu to close the currently selected file, or



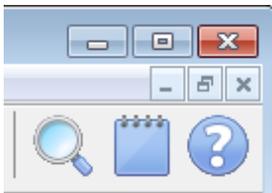
click *Close All* on the *Windows* menu which will close all currently open files.

The most often used way to close a file is to click the  button in the caption bar of the file's [Code Editor](#) window in the [Workspace](#). Shown in screenshot below.



The [Code Editor](#) Window has to be in either Normal or Minimized mode for the  button to be visible.

If the [Code Editor](#) window is maximized then the  button changes to a  button and moves to the right end of the [Main Menu](#) bar of the IDE, as shown below.



Clicking the  button in the [Caption Bar](#) of the IDE will cause all open files to close before the IDE is closed.

NOTE: Attempting to close any file with pending changes will result in prompting the User to determine whether the changes should be saved or not.

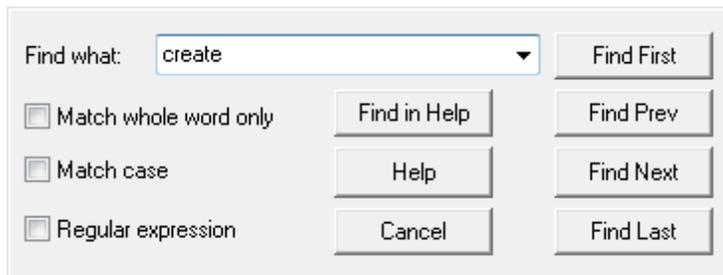
11.4.5 Find Text in a File

In order to find a word or phrase in a file a *search term* has to be defined. The *search term* is stored in an IDE global variable. This allows the User to search for a term in one [Code Editor](#) and then select a different [Code Editor](#) and search for the same term without having to re-enter it.

Each time the IDE is opened there is no established *search term*. In order to create a search term the User has to open the *Find* dialog (shown below and described in detail [here](#)).

The *Find* dialog can be opened by the User via one of the following methods provided there is at least one [Code Editor](#) window open:

- Pressing <CTRL>+F on the keyboard.
- Selecting the [Main Menu](#) *Editor/Find* option.
- Right-clicking in the [Code Editor](#) and selecting *Find* from the popup menu.



Each time the dialog is opened the *search term* in "Find what" will contain one of three possible values, described below.

1. Blank - If it is the first time the dialog has been opened during this session of the IDE.
2. Previous *search term* - the stored *search term* used in the last find operation.
3. New *search term* - Occurs when a word or phrase is highlighted in the currently selected [Code Editor](#) window when the *Find* dialog is opened.

The User may use the existing *search term*, if one exists, or enter a new *search term*.

All the buttons and options provided in the *Find* dialog are self-explanatory. Each is described [here](#) in detail.

However, it should be mentioned that the following buttons have shortcut keys associated with them.

- Find First <SHFT>+F2
- Find Prev F2
- Find Next F3
- Find Last <SHFT>+F3
- Find in Help..F1

The User may move freely between [Code Editor](#) windows when using the buttons and/or shortcut keys.

All Find Prev/Find Next find actions start at the current carat location in the currently selected [Code Editor](#) window.

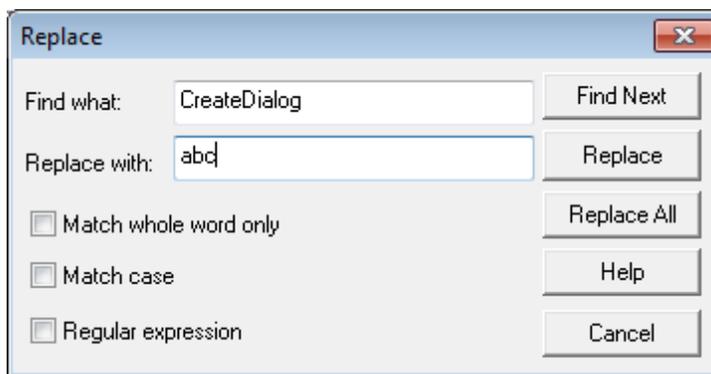
11.4.6 Replace Text in a File

In order to find and replace a word or phrase in a file a *search term* and a *replacement term* have to be defined. Both terms are stored in IDE global variables. This allows the User to search/replace in one [Code Editor](#) window and then select a different [Code Editor](#) window and search/replace with the same terms without having to re-enter them.

Each time the IDE is opened there are no established *search and replacement terms*. In order to create the terms the User has to open the *Replace* dialog (shown below and described in detail [here](#)).

The *Replace* dialog can be opened by the User via one of the following methods provided there is at least one [Code Editor](#) window open:

- Pressing <CTRL>+H on the keyboard.
- Selecting the [Main Menu](#) *Editor/Replace* option.
- Right-clicking in the [Code Editor](#) and selecting Replace from the popup menu.



Each time the dialog is opened the *search* and *replacement terms* will contain one of three possible pairs of values, described below.

1. Both Blank - If it is the first time the dialog has been opened during this session of the IDE.
2. Both Previous terms - the stored *terms* used in the last replace operation.
3. New *search term* / old *replacement term* - Occurs when a word or phrase is highlighted in the currently selected [Code Editor](#) window when the *Replace* dialog is opened.

The User may use the existing *terms*, if they exist, or enter a new *term* for *either or both*. All the buttons and options provided in the *Replace* dialog are self-explanatory. Each is described [here](#) in detail.

The User may move freely between [Code Editor](#) windows when using the buttons. All replace actions start at the current carat location in the currently selected [Code Editor](#) window.

11.5 Single File Application

This section describes how to perform the following actions on single file applications:

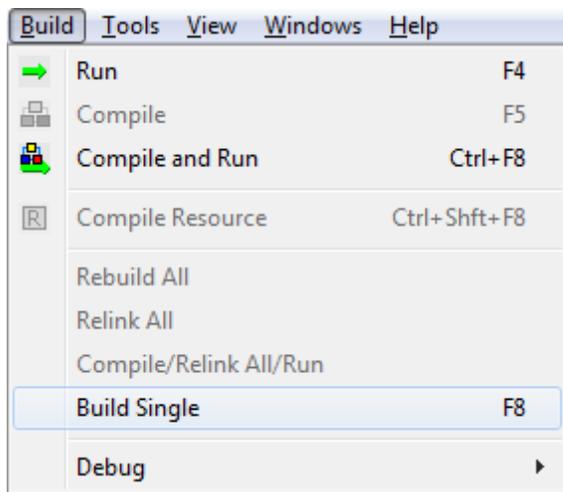
- [Set Application Options](#)
- [Compile an Application](#)

- [Run an Application](#)

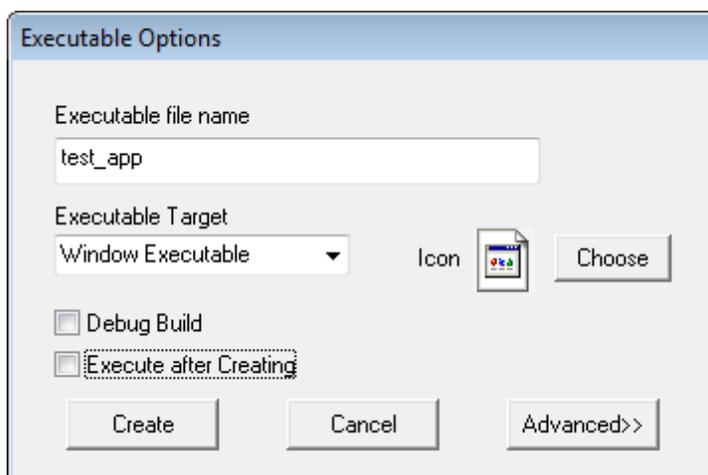
11.5.1 Set Application Options

A single source file can not be compiled into a useable file until the application's options are established. The options are stored in a binary file with the same base name as the source file and with a file extension of ".opts". The User can easily determine if the file exists, from inside the IDE, by looking at the [Main Toolbar](#). If the *Compile* button, , and the *Compile and Execute* button, , are disabled then the options file (for the currently selected source file in the [Workspace](#)) does not exist. The above assumes there is no project currently open.

The following describes the procedure to set the single file application options:



Selecting *Build / Build Single* from the [Main Menu](#), or clicking the  button, will open the *Executable Options* dialog, shown below. The dialog is used to set initial options or, subsequently, to modify options.



The dialog, as it appears at left, allows the User to set the basic options. Each item is described in the table below.

Item	Description
------	-------------

Executable File Name	When the dialog is opened for the first time, for a given source file (.iwb only), the name is automatically assigned by the dialog. The entry will be the same as the source file name. The user can use the default or edit the entry.
Executable Target	Combo box used to select the desired type of executable. Valid options are: <ul style="list-style-type: none"> • Windows Executable • Console Executable • Windows DLL
Icon	Displays the icon to be associated with this executable. The default (as shown above) is the first icon in the <i>windows/system32/shell32.dll</i> file. The selected icon will be added as a resource to the executable.
Icon Choose	Used to open the <i>Change Icon</i> dialog allowing the User to select a different icon and/or a different file to look in for the icon.
Debug Build	Checking this option will cause a binary file, with information used to debug the application, to be created. The file will be in the same folder as the source file. The name will be the same as the source file but with a <i>.dbgs</i> extension. The final application should be created with this option unchecked.
Execute after Creating	Checking this option will cause the application to be compile and run each time the <i>Application Options</i> dialog is closed.
Create	Saves any pending changes to the source file's associated option's file, which has the same name as the source file with an <i>.opts</i> file extension. If the Execute after Creating option is checked the application to be compile and run.
Cancel	Cancel's any pending changes and no other activity takes place.
Advanced>>	Expands the <i>Application Options</i> dialog (shown below) to display advanced options which are not routinely needed. They are covered in the table below.

The expanded *Application Options* dialog showing the advanced options. The advanced options are described in the table below.

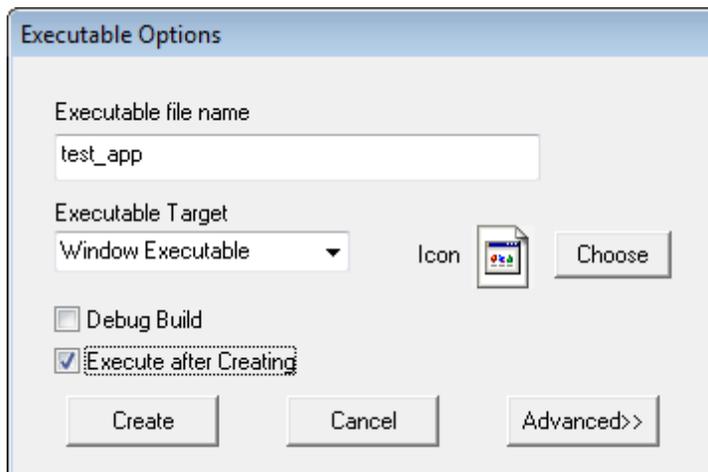
Note: To collapse the advanced section the User clicks the *Advanced<<* button.

Item	Description
Stack size	The maximum amount of memory for the storage of nested subroutines' local variables. The default is 0x100000 (1,048,576). If this number is exceeded the compiler will generate a "stack space" warning. The User then can increase this number.
Stack commit size	The maximum amount of memory for the storage of a single subroutine's local variables. The default is 32k (32,768). If a given subroutine exceeds this number the compiler will generate a "stack space" warning. The User then can increase this number.
Base address	Memory address to load DLL Default is 0x10000000. Any change should be in multiple of 64K.
Assembler options	Used to specify options to NASM assembler. See <i>Chapter 2: Running NASM</i> section of the <i>Help/ Assembler documentation</i> option in the Main Menu .
Include relocation	Used to allow DLL's to be loaded at other than the preferred address in memory. Can be used by advanced Users to load exe files dynamically.

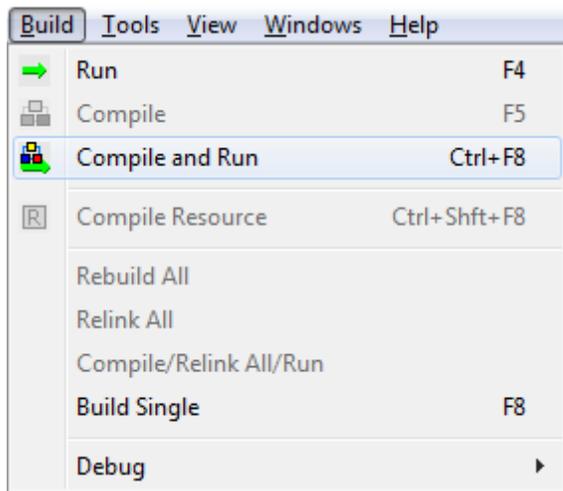
table in executables	
Generate Linker Map	Used to provide additional information to the User when debugging.

11.5.2 Compile an Application

The User has several options to compile a source file.



Clicking *Create* with the *Execute after Creating* option selected in the *Executable Options* dialog will compile the source file. If successful, the application will be executed.



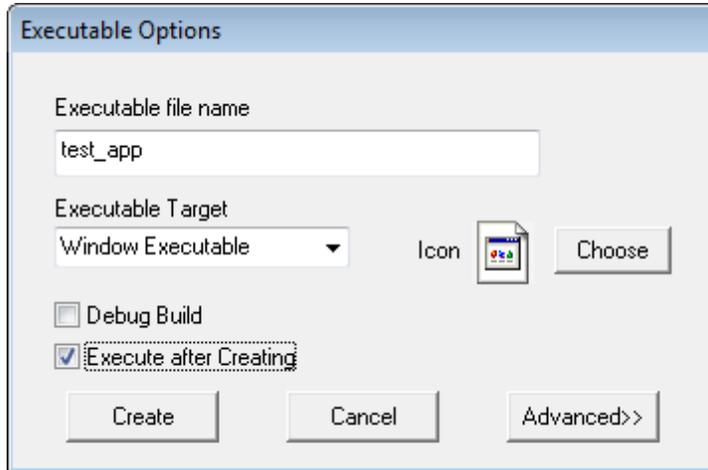
Selecting the *Build / Compile and Run* option from the [Main Menu](#) will have the same results as above. as will clicking the  button on the [Main Toolbar](#).

And finally, clicking the  button on the [Main Toolbar](#) will compile the source file only.

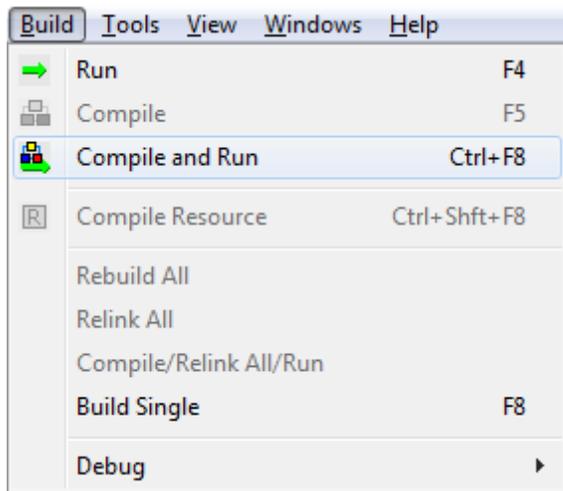
Note: When compiling, messages will appear in the *Build Tab* of the [Output Window](#).

11.5.3 Run an Application

The User has several options to execute an application.



Clicking *Create* with the *Execute after Creating* option selected in the *Executable Options* dialog will compile the source file. If successful, the application will be executed.

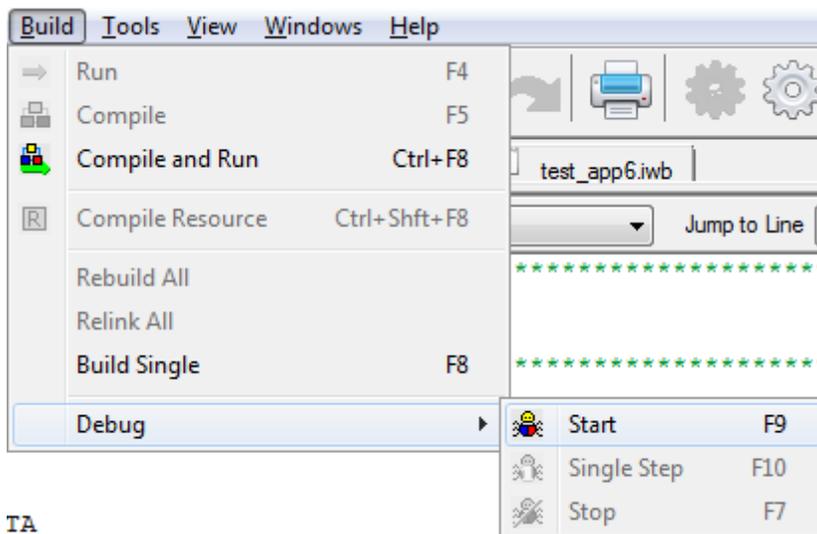


Selecting the *Build / Compile and Run* option from the [Main Menu](#) will have the same results as above. as will clicking the  button on the [Main Toolbar](#).

Clicking the  button on the [Main Toolbar](#) will execute the application.

NOTE: The following *Debug* option is not currently available.

If the *Debug* option is selected in the *Executable Options* dialog then the following methods are available:



Selecting the *Build / Debug / Start* option from the [Main Menu](#) will execute the application. Clicking the  button on the [Main Toolbar](#) will do the same, as well as pressing the <F9> key.

TA

11.6 Projects

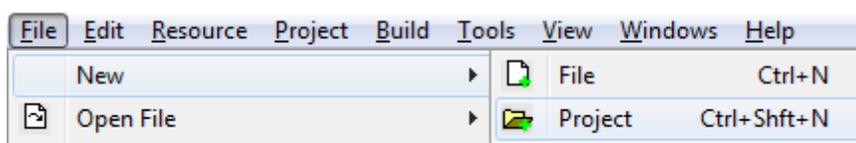
This section describes how to perform the following actions on single file applications:

- [Create a New Project](#)
- [Set Project Options](#)
- [Open a Project](#)
- [Save a Project](#)
- [Close a Project](#)
- [Add Files](#)
- [Remove Files](#)
- [Compile a Project](#)
- [Run a Project](#)
- [Resources](#)
- [Compile a Resource](#)

11.6.1 Create a New Project

In order to create an application based upon multiple source files the User needs to create a *Project* file. A *Project* file is a binary file that contains almost the same information as the *.opts* file used by [Single File Applications](#). Additionally, the file will contain the file names of all the source files that comprise the project. The *Project* file will have an *.iwp* file extension.

The following describes the procedure to create a Project file:



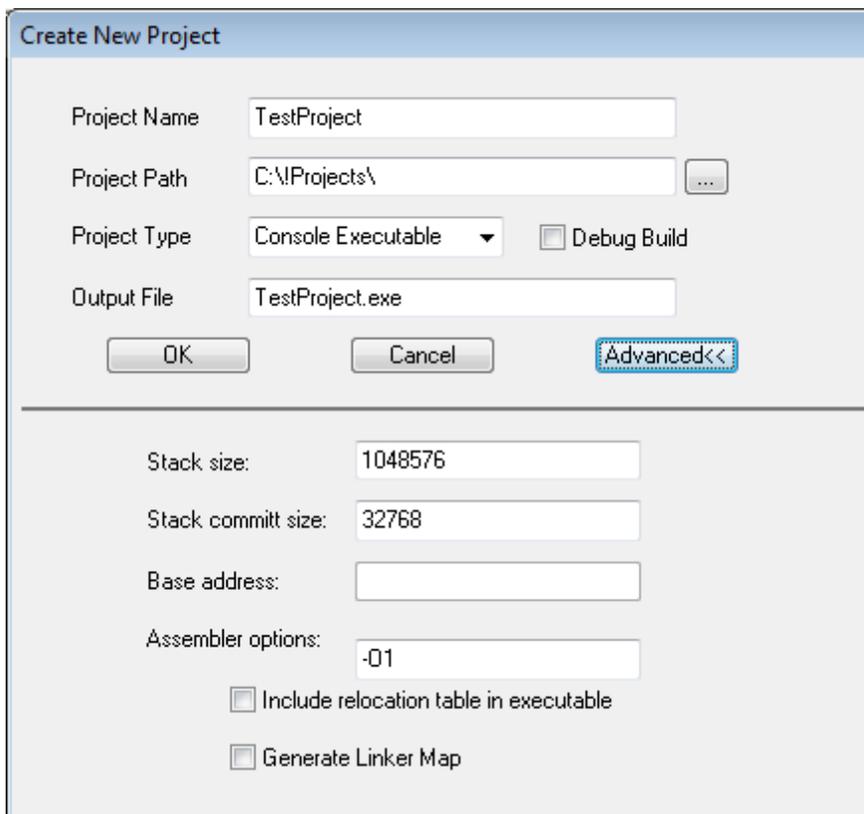
Selecting *File / New Project* from the [Main Menu](#), or clicking the  button on the [Main Toolbar](#), will open the

Create New Project dialog, shown below.

The *Create New Project* dialog is used to create a project and set its initial options. The dialog, as it appears at left, allows the User to set the basic options. Each item is described in the table below.

Item	Description
Project Name	The User selected unique name for the project
Project Path	Path to folder where project file will be created. When the dialog is opened the default is the last folder that was used to create a new project. The button to the right is used to open a dialog to browse to a different folder.
Project Type	Combo box used to select the desired type of executable. Valid options are: <ul style="list-style-type: none"> • Windows Executable - if the program will use any windows, dialogs, controls, etc. • Console Executable - for a text only console program. • Windows DLL - to create a DLL project. • No Output - for a compile-only project. • Static Library - to create a static library project
Debug Build	Checking this option will cause binary files, with information used to debug the application, to be created. The final application should be created with this option unchecked.
Output File	The name of the application executable. This entry will track any changes entered in the <i>Project Name</i> entry. If the User wants the application to have a name different than the <i>Project Name</i> then the <i>Output File</i> name will have to be entered after all changes to the <i>Project Name</i> have been made. As any changes are made to the <i>Project Type</i> the file extension of the <i>Output File</i> will change accordingly.
OK	Causes the following sequence of events: <ul style="list-style-type: none"> • Creates a project file named <i>Project Name</i> + ".iwp" in the <i>Project Path</i>

	<p>folder.</p> <ul style="list-style-type: none"> • Saves the entered information to the project file. • Closes the current project in the IDE, if one is opened. The User is prompted to save any pending changes to the current project, if they exists. • Loads the new project into the IDE, making it the current project.
Cancel	Cancel's the process and no project is created..
Advanced>>	Expands the <i>Create New Project</i> dialog (shown below) to display advanced options which are not routinely needed. They are covered in the table below.



The expanded *Application Options* dialog showing the advanced options. The advanced options are described in the table below.

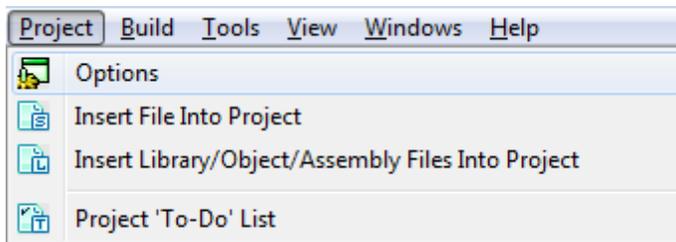
Note: To collapse the advanced section the User clicks the *Advanced<<* button.

Item	Description
Stack size	The maximum amount of memory for the storage of nested subroutines' local variables. The default is 0x100000 (1,048,576). If this number is exceeded the compiler will generate a "stack space" warning. The User then can increase this number.
Stack commit	The maximum amount of memory for the storage of a single subroutine's local variables. The default is 32k (32,768). If a given subroutine exceeds this number

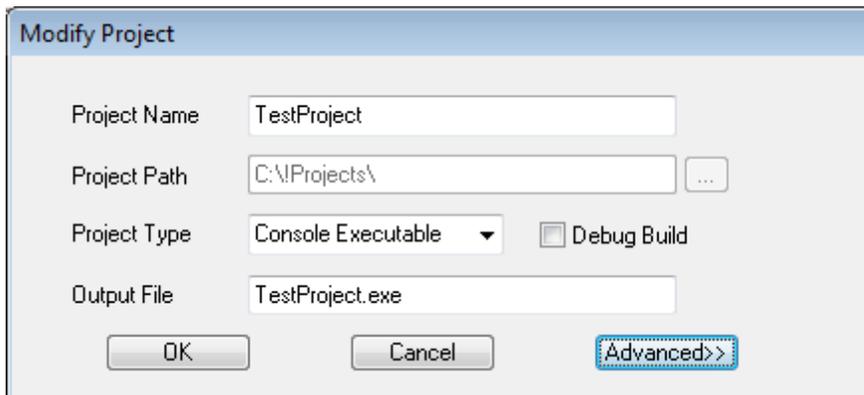
size	the compiler will generate a "stack space" warning. The User then can increase this number.
Base address	Memory address to load DLL. Default is 0x10000000. Any change should be in multiple of 64K.
Assembler options	Used to specify options to NASM assembler. See <i>Chapter 2: Running NASM</i> section of the <i>Help/ Assembler documentation</i> option in the Main Menu .
Include relocation table in executables	Used to allow DLL's to be loaded at other than the preferred address in memory. Can be used by advanced Users to load exe files dynamically.
Generate Linker Map	Used to provide additional information to the User when debugging.

11.6.2 Set Project Options

The following describes the procedure to modify the current project's options.



Selecting *Project / Options* from the [Main Menu](#) will open the *Modify Project* dialog, shown below.



The *Modify Project* dialog is used to modify a project's options. The dialog, as it appears at left, allows the User to modify the basic options. Each item is described in the table below.

Item	Description
------	-------------

Project Name	The User selected unique name for the project
Project Path	The project path can not be changed once it is established when the project is created. The entry is always disabled here.
Project Type	Combo box used to select the desired type of executable. Valid options are: <ul style="list-style-type: none"> • Windows Executable - if the program will use any windows, dialogs, controls, etc. • Console Executable - for a text only console program. • Windows DLL - to create a DLL project. • No Output - for a compile-only project. • Static Library - to create a static library project
Debug Build	Checking this option will cause binary files, with information used to debug the application, to be created. The final application should be created with this option unchecked.
Output File	The name of the application executable. This entry will track any changes entered in the <i>Project Name</i> entry. If the User wants the application to have a name different than the <i>Project Name</i> then the <i>Output File</i> name will have to be entered after all changes to the <i>Project Name</i> have been made. As any changes are made to the <i>Project Type</i> the file extension of the <i>Output File</i> will change accordingly.
OK	Causes any pending changes to the current project.
Cancel	Cancel's any pending changes...
Advanced>>	Expands the <i>Modify Project</i> dialog (shown below) to display advanced options which are not routinely needed. They are covered in the table below.

The screenshot shows the 'Modify Project' dialog box with the following fields and options:

- Project Name: TestProject
- Project Path: C:\Projects\
- Project Type: Console Executable (dropdown menu)
- Output File: TestProject.exe
- Buttons: OK, Cancel, Advanced<<
- Stack size: 1048576
- Stack committ size: 32768
- Base address: (empty field)
- Assembler options: -O1
- Include relocation table in executable:
- Generate Linker Map:

The expanded *Modify Project* dialog showing the advanced options. The advanced options are described in the table below.

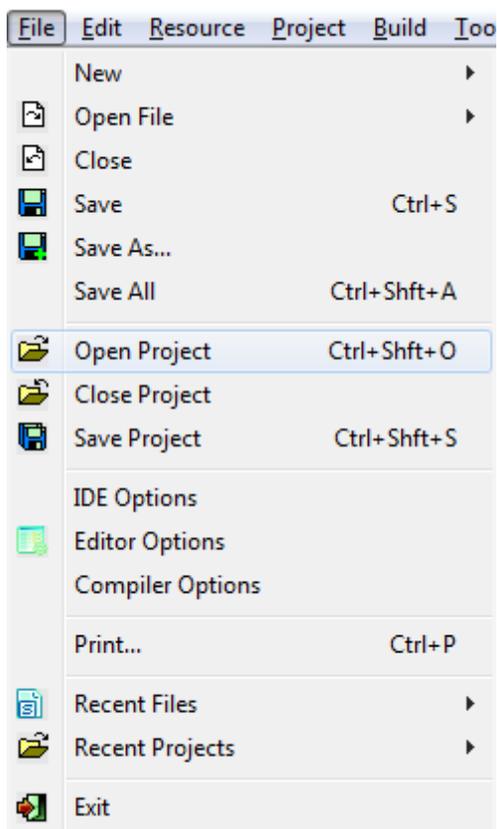
Note: To collapse the advanced section the User clicks the *Advanced<<* button.

Item	Description
Stack size	The maximum amount of memory for the storage of nested subroutines' local variables. The default is 0x100000 (1,048,576). If this number is exceeded the compiler will generate a "stack space" warning. The User then can increase this number.
Stack commit size	The maximum amount of memory for the storage of a single subroutine's local variables. The default is 32k (32,768). If a given subroutine exceeds this number the compiler will generate a "stack space" warning. The User then can increase this number.
Base address	Memory address to load DLL. Default is 0x10000000. Any change should be in multiple of 64K.
Assembler options	Used to specify options to NASM assembler. See <i>Chapter 2: Running NASM</i> section of the <i>Help/Assembler documentation</i> option in the Main Menu .
Include relocation table in	Used to allow DLL's to be loaded at other than the preferred address in memory. Can be used by advanced Users to load exe files dynamically.

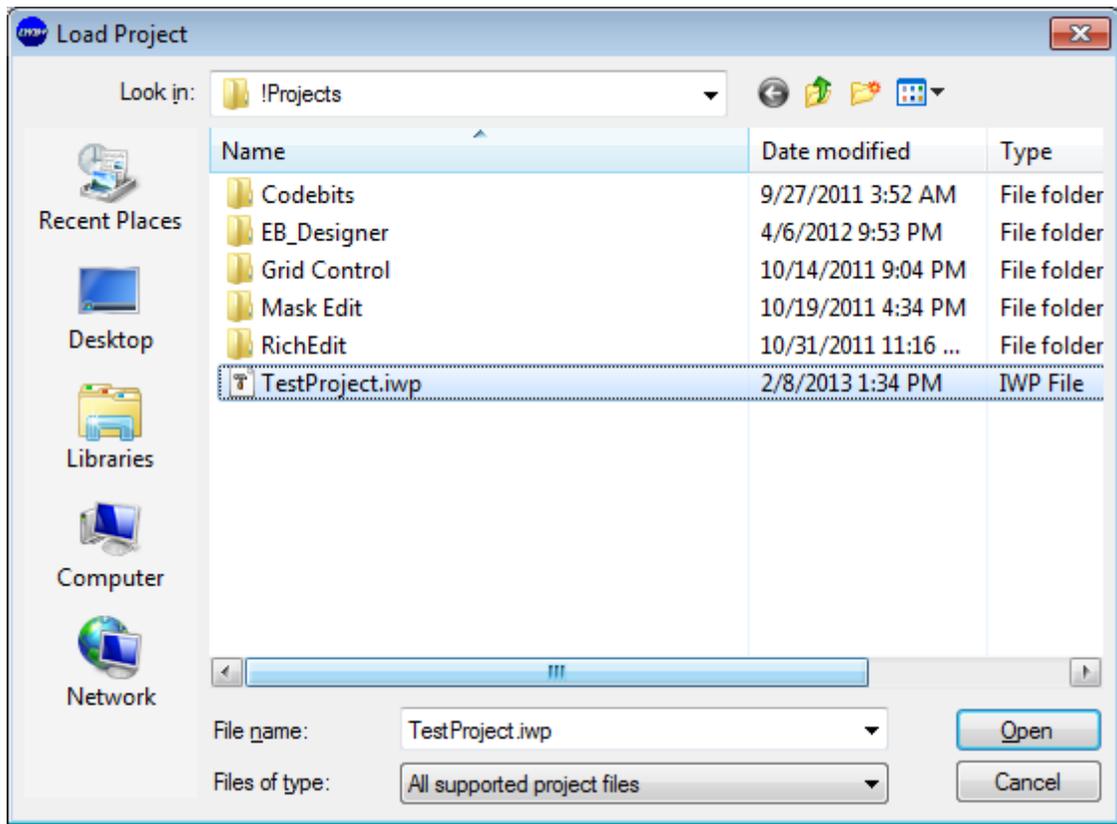
executables	
Generate Linker Map	Used to provide additional information to the User when debugging.

11.6.3 Open a Project

There are several ways the User can open a project and load it into the IDE.



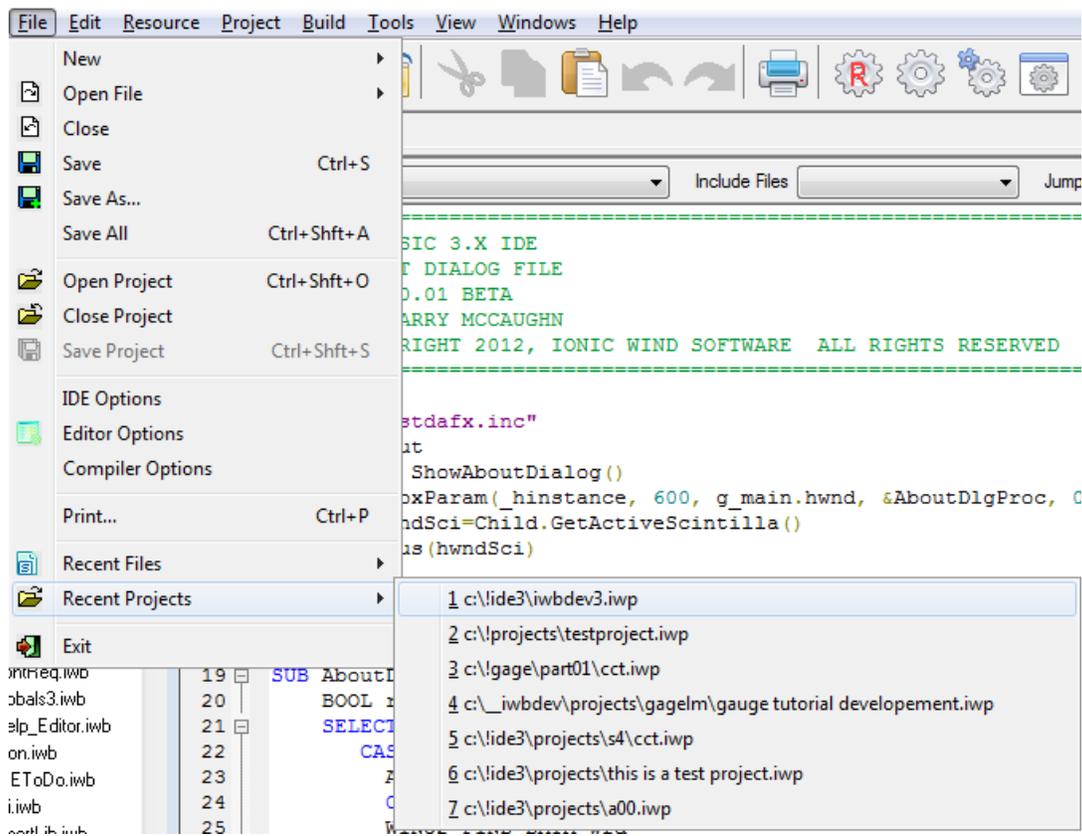
Clicking *File / Open Project* option from the [Main Menu](#) or clicking the  button on the [Main Toolbar](#) will open the *Load Project* dialog, shown below.



The Load Project dialog allows the user to select an existing project file, *.iwp. For backwards compatibility, *.ebp files may also be loaded. When a project

is
sele
cte
d
and
*Op
en*
is
clic
ked
the
proj
ect
will
be
loa
ded
and
bec
om
e
the
curr
ent
proj
ect
.

The User also has the option of selecting a recently loaded project to be reloaded.



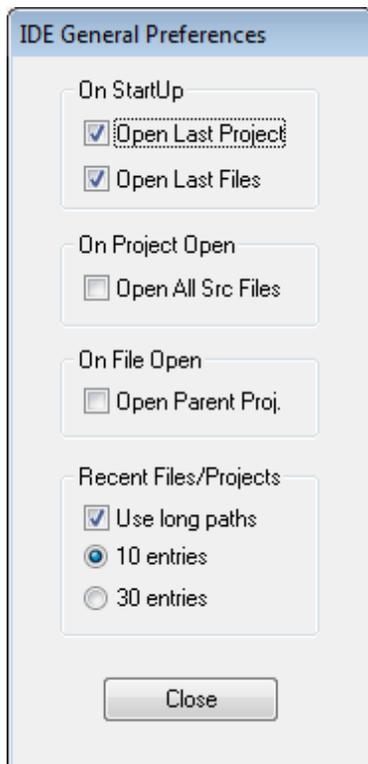
Clicking *File / Recent Projects* option from the [Main Menu](#) opens a sub-menu with a list of recent projects. The entries are listed most recent to least recent.

Double clicking an entry will cause the project to be loaded and become the current project.

If the selected project

has been moved or deleted, the User will be notified, the entry removed, and no project is loaded.

Note: If there is a currently opened project with pending changes when another project is opened the User will be prompted to save the changes prior to closing the current project.



The last opened project will be automatically reopened on IDE startup if the *On Startup / Open Last Project* option is selected in the *IDE General Preferences* dialog.

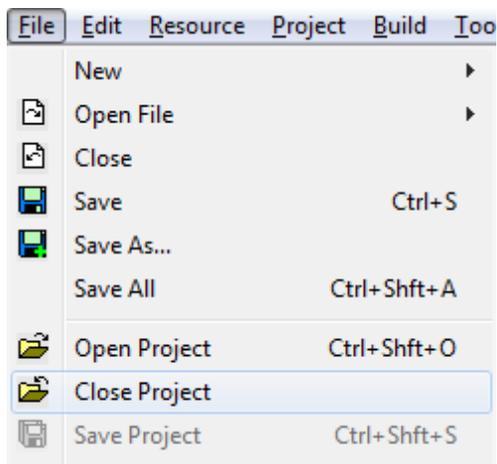
See the [How-To»Set Startup Preferences](#) section for additional information.

If the project directory was renamed, or copied from a different source, you will be prompted to update the project files. Selecting "Yes" to the prompt will adjust all of the internal paths in the project file to the new directory. The resource file will also be scanned for any needed path updates.

11.6.4 Close a Project

The current project can be closed at any time by selecting the File / Close Project option from the Main Menu or clicking the button on the Main Toolbar.

If there are any pending changes to the project file the User will be prompted to save them.



Note: This applies only to the project file (*.iwp) itself and not to any files that are part of the project.

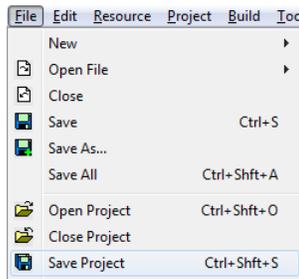
11.6.5 Save a Project

When changes are made to the current project file the User can save the changes by selecting the File / Save Project option, shown below, in the [Main Menu](#).

The option will be disabled when there are no pending changes.

Note: This applies only to the project file (*.iwp) itself and not to any files that are part of the

project.



11.6.6 Add Files

Once a project has been created and loaded into the IDE, the User can begin to add source files to the project.

A project can have as many files as the User desires added to it. However, there are two conditions that must be met:

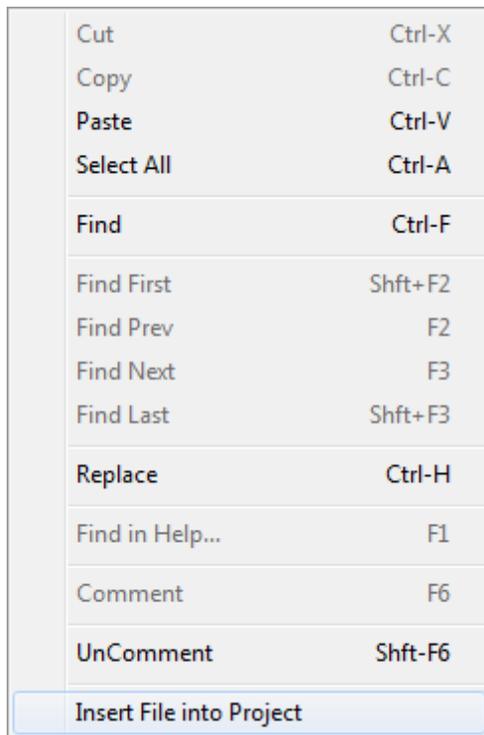
1. Every project must have at least one Source file.
2. Every Windows or Console Executable project must contain a *\$MAIN* directive to indicate where execution should begin.

There are five types of files that the User can add to a project; [*.iwb](#), [*.obj](#), [*.asm](#), [*.res](#), and [*.lib](#). Each is covered in its own section below.

****.iwb files***

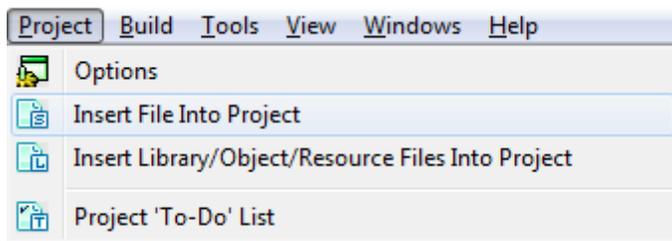
This section also covers **.eba* files for backward compatibility.

1. Open an existing IWBasic source file or create a new one by following one of the procedures described in the [How-To»Files](#) section.
2. Right click the file's [Code Editor](#) window and select *Insert File into Project* from the popup menu (shown below). Alternately, the User can select the *Project / Insert File into Project* option from the [Main Menu](#) (shown below).
3. The added file will be listed in the *File* tab of the [Project List](#) window
4. Select the *File / Save Project* option from the [Main Menu](#) to make the addition permanent. See [Note 1](#) below.



Right-click popup menu used to add *.iwb, *.eba, *.asm, and *.a files to a project.

NOTE: The *Insert File into Project* option is not available for new file that has never been saved. Once the file has been saved with its permanent name the option will be available.



[Main Menu](#) options for adding all allowed file types to a project.

***.obj files**

This section also covers *.o files.

A project may contain external object files. These can be specified in the source code with the [\\$USE](#) statement or added to the project by following these steps:

1. Select *Project / Insert Library/Object/Resource Files Into Project* option from the [Main Menu](#).
2. Select one or more object files to insert from the *File* dialog and press *Open*.
3. The added file(s) will be listed in the *File* tab of the [Project List](#) window
4. Select the *File / Save Project* option from the [Main Menu](#) to make the addition permanent. See [Note 2](#) below.

***.asm files**

This section also covers *.a files. Follow the same steps as outlined in the [*.iwb](#) section above. See [Note 1](#) below.

****.res files***

In addition to the primary resource file described in the [Resources](#) section the User can add additional resource files to the project by following the steps in the [*.obj](#) section above.

****.lib files***

A project may contain external library files. These can be specified in the source code with the [\\$USE](#) statement or added to the project by following these steps:

5. Select *Project / Insert Library/Object/Resource Files Into Project* option from the [Main Menu](#).
6. Select one or more library files to insert from the *File* dialog and press *Open*.
7. The added file(s) will be listed in the *File* tab of the [Project List](#) window
8. Select the *File / Save Project* option from the [Main Menu](#) to make the addition permanent. See [Note 2](#) below.

Note 1

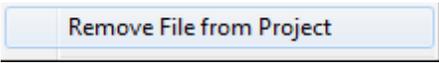
It's common practice to keep all of the source files used in a project in the project directory. However, this is not necessary and your source files can be located anywhere on your drive.

Note 2

For a static library project the external object files must be in the same directory as the project output. For other project types the external library/object can be located anywhere on the drive.

11.6.7 Remove Files

If a project file is no longer needed by the project the User can remove it by right-clicking on the file name in the list under the *File* tab of the *Project List* window and choosing *Remove File from Project* option in the popup menu, shown below.



Removing a source file from a project does not delete the file, it just removes it from the list of source files in the project. The User can safely re-add the file if necessary.

Select the *File / Save Project* option from the [Main Menu](#) to make the deletion permanent.

11.6.8 Compile a Project

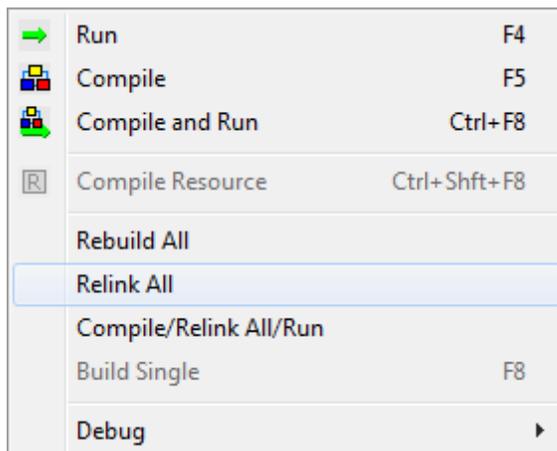
After adding source files and resources, if any, to the project the User can build the executable or DLL by one of the methods described in this section.

The normal sequence of events when compiling a project are:

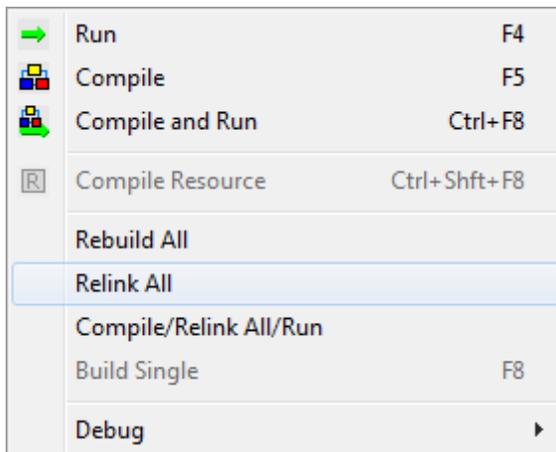
1. Compile the primary resource file if any resources are used
2. Compile all the project source file to *.o files
3. Link all the compiled object files and libraries
4. Create executable file

As a project is being built progress will be displayed in the *Build* tab of the [Output Window](#)(see for additional information).

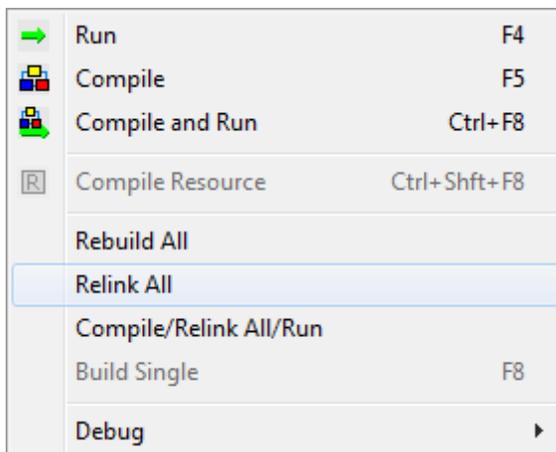
Note: If the [Output Window](#) is not open and the *Build* tab selected when a build process is started, it will automatically be done so.



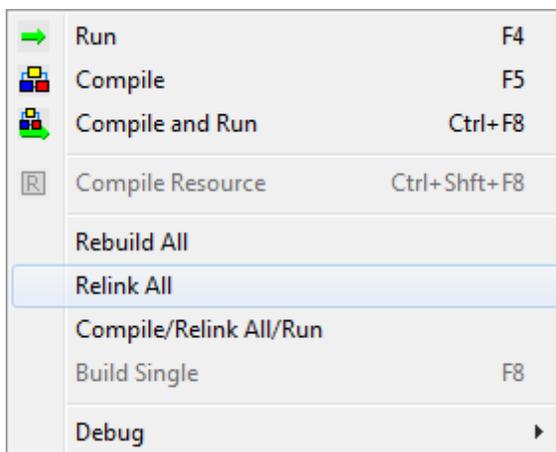
Selecting *Build / Rebuild All* from the [Main Menu](#) will result in all four steps above being executed. This is the same as clicking the  button on the [Main Toolbar](#).



For projects with a large number of source files there are times when only one file is edited. Selecting *Build / Compile* from the [Main Menu](#) will compile just the currently selected source file in the [Workspace](#). Step 2 from the list above will be performed for the single file.
Note: See next entry.



After compiling a single file (described above) the User can select *Build / Relink All* from the [Main Menu](#) to perform steps 3-4 above.



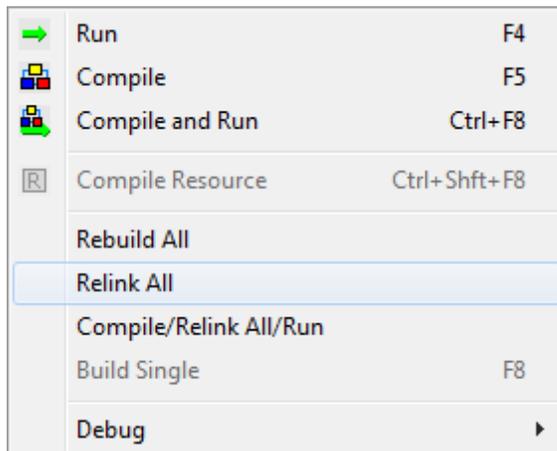
For projects with a large number of source files there are times when only one file is edited. Selecting *Build / Compile/Relink All/Run* from the [Main Menu](#) will compile just the currently selected source file in the [Workspace](#). If the compile is successful all the object files in the project will be linked to form an executable. If the exe file is created successfully then the application will be executed.

Note: Use of this menu option requires that the entire project be previously compiled so that all the project's object files exist.

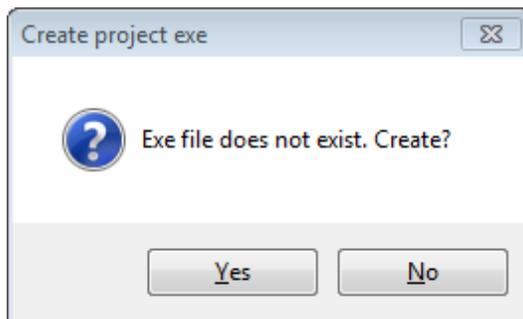
There are also several options available to the User which are combinations of the four steps above and running the executable.

Clicking the  button will cause the project to compile (the four steps above) and, if the build is

successful, the program to execute.

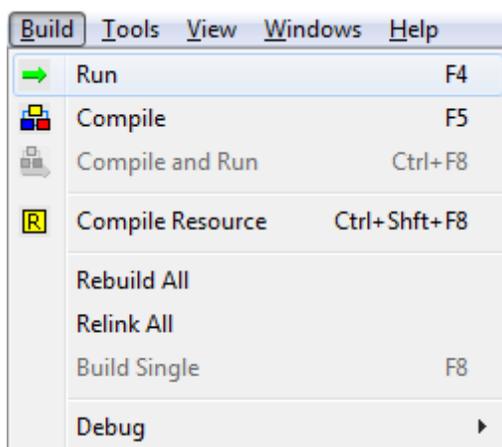


If *Build / Run* is selected from the [Main Menu](#) (or the  button is clicked on the [Main Toolbar](#)) and there is no executable file, the User will be asked if an executable should be created via the dialog below.

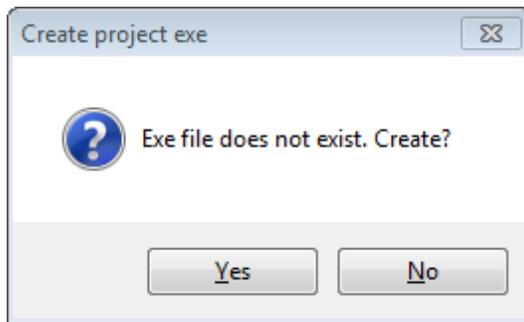


Clicking *Yes* will result in all four steps above being executed and then the program being executed.

11.6.9 Run a Project



Selecting *Build / Run* from the [Main Menu](#) or clicking the  button on the [Main Toolbar](#) will cause the executable to run. If there is no executable file, the User will be asked if an executable should be created via the dialog below.

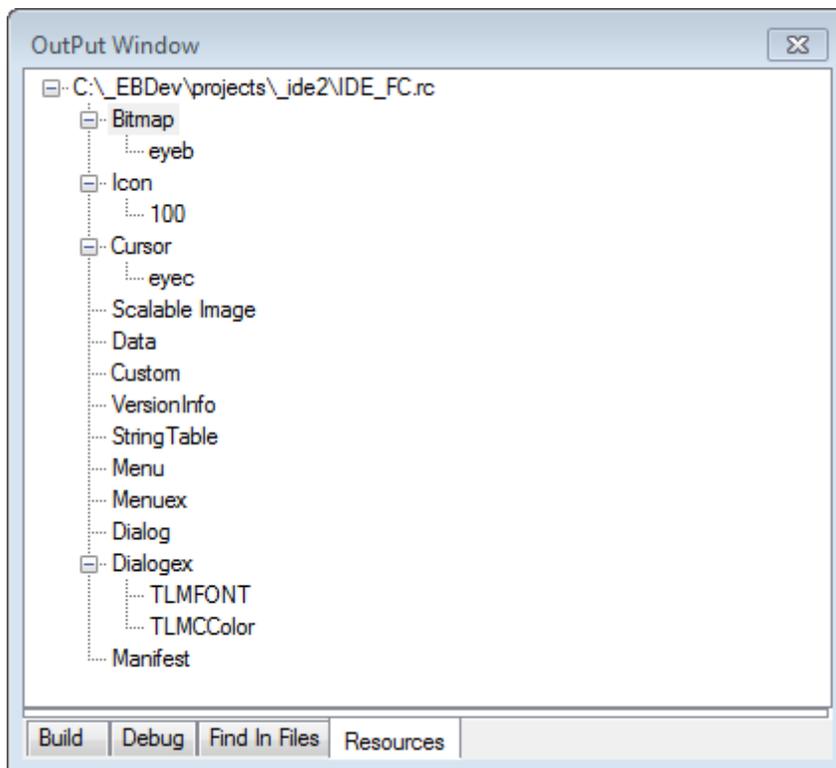


Clicking *Yes* will result in the project being rebuilt (see the [How-To»Projects» Compile a Project](#) section for additional information) and then the program will be executed.

11.6.10 Resources

This section covers the adding, deleting, and editing of resources in a project.

Note: This section assumes the User has read the section covering the [Resources](#) tab of the [Output Window](#) and the [General Programming»Using Resources](#) section.



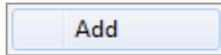
The [Resources](#) tab of the [Output Window](#), shown at left, is where all resource activities take place. Each available type of resource is listed and resources added by the User are listed under the corresponding type heading.

In order for the User to modify resources the [Resources](#) tab of the [Output Window](#) has to be opened by one of the following methods:

1. Select the [Resources](#) tab of the [Output Window](#). If the window is closed then select the *View / Output Window* from the [Main Menu](#) and then select the [Resources](#) tab.
2. Select *Resources / Add* from the [Main Menu](#) which will automatically open the [Output Window](#), if closed, and automatically select the [Resources](#) tab.

Adding Resources

The User can add a resource by right-clicking the desired resource type and then selecting *Add* from the popup menu, shown below.



The *Resource Editor* dialog, shown below, will open.



The *Resource ID* is a suggested ID consisting of a three letter prefix followed by a number. The User may change it as long as it is unique to the application.

Each of the other fields may or may not be enabled depending upon the resource type.

Save will save any pending changes while *Cancel* discards any pending changes.

The entries for each of the available resource types follows below.

Bitmap

<i>Resource ID</i>	BMPx
<i>Resource Type</i>	BITMAP, disabled
<i>File</i>	Contains the full pathname of the desired *.bmp file. Use the associated button to navigate to file.
<i>Content</i>	"" , disabled

Note:	When using the file dialog to navigate to desired file the User may select multiple files. Each will be added and the <i>Resource ID's</i> for each will be sequential starting with the <i>Resource ID</i> shown in the <i>Resource Dialog</i> .
-------	---

Icon

<i>Resource ID</i>	ICOx
<i>Resource Type</i>	ICON, disabled
<i>File</i>	Contains the full pathname of the desired *.ico file. Use the associated button to navigate to file.
<i>Content</i>	"" , disabled
Note:	When using the file dialog to navigate to desired file the User may select multiple files. Each will be added and the <i>Resource ID's</i> for each will be sequential starting with the <i>Resource ID</i> shown in the <i>Resource Dialog</i> .

Cursor

<i>Resource ID</i>	CURx
<i>Resource Type</i>	CURSOR, disabled
<i>File</i>	Contains the full pathname of the desired *.cur file. Use the associated button to navigate to file.
<i>Content</i>	"" , disabled
Note:	When using the file dialog to navigate to desired file the User may select multiple files. Each will be added and the <i>Resource ID's</i> for each will be sequential starting with the <i>Resource ID</i> shown in the <i>Resource Dialog</i>

Scalable Image

<i>Resource ID</i>	IMGx
<i>Resource Type</i>	RTIMAGE, disabled
<i>File</i>	Contains the full pathname of the desired *.jpg;*.gif; or *.bmp file. Use the associated button to navigate to file.
<i>Content</i>	"" , disabled
Note:	When using the file dialog to navigate to desired file the User may select multiple files. Each will be added and the <i>Resource ID's</i> for each will be sequential starting with the <i>Resource ID</i> shown in the <i>Resource Dialog</i>

Data

<i>Resource ID</i>	DATx
<i>Resource Type</i>	RCDATA, disabled
<i>File</i>	"" , disabled
<i>Content</i>	BEGIN ;Enter data here END
Note:	Select <i>Help / Assembler</i> docs from the Main Menu for details on coding the <i>Content</i> field.

Custom

<i>Resource ID</i>	CUSx
<i>Resource Type</i>	333, enabled
<i>File</i>	Contains the full pathname of the desired *.jpg;*.gif, or *.bmp file. Use the associated button to navigate to file.
<i>Content</i>	"" , disabled
Note:	<i>Resource Type</i> can be any User desired number above 256.

VersionInfo

<i>Resource ID</i>	VERx
<i>Resource Type</i>	VERSIONINFO, disabled
<i>File</i>	"" , disabled
<i>Content</i>	<pre> FILEVERSION 0,0,0,0 PRODUCTVERSION 0,0,0,0 FILEFLAGSMASK 0x00000000L FILEFLAGS 0x00000000L FILEOS 0x00000000L FILETYPE 0x00000000L FILESUBTYPE 0x00000000L BEGIN BLOCK "StringFileInfo" BEGIN BLOCK "040904E4" BEGIN VALUE "CompanyName", "My Company." VALUE "Contact e-mail", "you@company.com" VALUE "FileDescription", "MySoftware.Exe" VALUE "FileVersion", "1.00" </pre>

	<pre> VALUE "Development Language", "IWBasic" VALUE "LegalCopyright", "Copyright© 2006" END END BLOCK "VarFileInfo" BEGIN VALUE "Translation", 0x0409, 1252 END END </pre>
Note:	Select <i>Help / Assembler docs</i> from the Main Menu for details on coding the <i>Content</i> field.

StringTable

<i>Resource ID</i>	STRx
<i>Resource Type</i>	STRINGTABLE, disabled
<i>File</i>	"", disabled
<i>Content</i>	<pre> BEGIN ;Enter data here END </pre>
Note:	Select <i>Help / Assembler docs</i> from the Main Menu for details on coding the <i>Content</i> field.

Menu

<i>Resource ID</i>	MNUx
<i>Resource Type</i>	MENU, disabled
<i>File</i>	"", disabled
<i>Content</i>	<pre> BEGIN ;Enter data here END </pre>
Note:	Select <i>Help / Assembler docs</i> from the Main Menu for details on coding the <i>Content</i> field.

Menuex

<i>Resource ID</i>	MNXx
<i>Resource Type</i>	MENUEX, disabled
<i>File</i>	"", disabled

<i>Content</i>	BEGIN ;Enter data here END
Note:	Select <i>Help / Assembler docs</i> from the Main Menu for details on coding the <i>Content</i> field.

Dialog

<i>Resource ID</i>	DLGx
<i>Resource Type</i>	DIALOG, disabled
<i>File</i>	The dialog coordinates in l, t, w, h format.
<i>Content</i>	BEGIN ;Enter data here END
Note:	Select <i>Help / Assembler docs</i> from the Main Menu for details on coding the <i>Content</i> field.

Dialogex

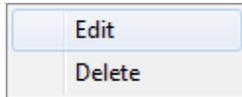
<i>Resource ID</i>	DLXx
<i>Resource Type</i>	DIALOGEX, disabled
<i>File</i>	The dialog coordinates in l, t, w, h format.
<i>Content</i>	BEGIN ;Enter data here END
Note:	Select <i>Help / Assembler docs</i> from the Main Menu for details on coding the <i>Content</i> field.

Manifest

<i>Resource ID</i>	MANx
<i>Resource Type</i>	MANIFEST, disabled
<i>File</i>	Contains the full pathname of the desired *.manifest file. Use the associated button to navigate to file.
<i>Content</i>	"" , disabled
Note:	Select <i>Help / Assembler docs</i> from the Main Menu for details on coding the <i>Content</i> field.

Editing Resources

The User can edit any resource entry by double-clicking the desired resource or by right clicking on the desired resource and selecting *Edit* from the popup menu, shown below.



Either action will open the *Resource Editor* dialog, shown above. The User is free to modify any enabled entry.

Any edits can be saved by clicking the *Save* button in the *Resource Editor* dialog

Deleting Resources

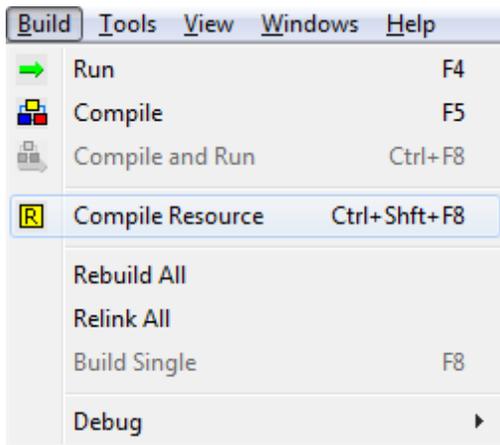
The User can delete any resource entry by right clicking on the desired resource and selecting *Delete* from the popup menu, shown above. The User will be asked to confirm the selection. Only the entry in the resource file will be deleted. Any associated external file will remain untouched and may be re-added a later time.

11.6.11 Compile a Resource

Normally, primary resource files are compiled at the beginning of the project build process. (See the [How-To»Projects»Compile a Project](#) section for details.)

When making numerous resource edits in a project that has a large number of source files it may save time to first resolve any resource file errors before invoking the entire build process.

This can be accomplished by selecting *Build / Compile Resource* from the [Main Menu](#) (shown below) or clicking the  button on the [Main Toolbar](#). The resource file alone will be compiled and the success or failure (with indicated errors) will be reported in the *Build* tab of the [Output Window](#).



11.7 Convert from CBasic / IBasic

Why convert to IWBASIC?

- To switch from an interpreted to compiled program.
- To use some of the advanced capabilities.
- To keep from re-writing the whole program.

Getting Started

The process is not as easy as it might seem. Some code translates easily – other code might be troublesome.

The important point is to follow a systematic approach in doing the conversion. By using the following as a checklist the process will be greatly simplified.

NOTE: For the rest of this discussion CB will be used to denote CBasic and IBasic code. IWB will be used to denote IWBASIC code.

First, create a new IWB file. Then copy and paste the desired CB code into it. Don't even try to compile the code at this point. It will fail.

Subroutines - Local

This section covers local, User coded subroutines / functions. It excludes calls to subroutines in components and calls to external subroutines in DLLs. Those will be covered later.

In CB, the RETURN statement indicates the end of a subroutine

```
SUB mysub1
  IF x = 4 THEN RETURN
  ...
RETURN
```

In IWB, the END SUB or ENDSUB statement indicates the end of a subroutine.

```
SUB mysub1
```

```

IF x = 4 THEN RETURN
...
RETURN
ENDSUB

```

Therefore, END SUB or ENDSUB has to be added after the last RETURN in each and every subroutine/function.

In CB, a name followed by a : can be used to define a subroutine.

```

mysub:
...
RETURN

```

In IWB, mysub: would indicate a LABEL. Therefore, it would need to be converted to:

```

SUB mysub
...
RETURN
ENDSUB

```

In CB, a subroutine with no passed parameters or return value can be called via the GOSUB command

```

GOSUB mysub1

SUB mysub1
...
RETURN

```

In IWB, it can be called exactly the same way

```

GOSUB mysub1

SUB mysub1
...
RETURN
ENDSUB

```

Or, it can be called like the following:

```

mysub1 ()

SUB mysub1
...
RETURN
ENDSUB

```

In CB, a subroutine that has no passed parameters but returns a value may appear like this:

```

a = 1
b = 2
PRINT mysub ()

SUB mysub
DEF c: INT
c = a + b
RETURN c

```

In IWB, the code would be:

```
a = 1
b = 2
PRINT mysub()

SUB mysub(), INT
DEF c: INT
c = a + b
RETURN c
ENDSUB
```

In CB, subroutines that have passed parameters are called functions, regardless of whether or not they return a value.

Functions have to be declared before they can be used.

```
DECLARE docircle(x:INT,y:INT,size:INT,colora:INT,colorb:INT)
...
docircle 100,100,25,color1,color2
...
SUB docircle(x,y,size,colora,colorb)
  CIRCLE win,x,y,size,colora,colorb
RETURN
```

In IWB the DECLARE and SUB statements are combined. This represents how all User defined subroutines, that don't return a value, have to be defined.

```
...
docircle (100,100,25,color1,color2)
...
sub docircle(x:INT,y:INT,size:INT,colora:INT,colorb:INT)
  CIRCLE win,x,y,size,colora,colorb
RETURN
ENDSUB
```

In CB, there is no requirement to pre-define the return value type.. The following are valid;

```
DECLARE mysub1(x: INT, y: INT)
DECLARE mysub2(x: STRING, y: STRING)
...
a=1
b=2
PRINT mysub1(a, b)

c = "My dog "
d = "has fleas"
PRINT mysub2(c, d)

SUB mysub1(x, y)
  z: INT
  z=x + y
RETURN z

SUB mysub2(x, y)
  z: STRING
  z=x + y
RETURN z
```

In IWB, the code will look like this:

```

a=1
b=2
PRINT mysub1(a, b)

c = "My dog "
d = "has fleas"
PRINT mysub2(c, d)

SUB mysub1(x: INT, y: INT), INT
  z: INT
  z=x + y
RETURN z
ENDSUB

SUB mysub2(x: STRING, y: STRING), STRING
  z: STRING
  z=x + y
RETURN z
ENDSUB

```

Subroutines - Components

In CB, component files allow the User to add common subroutines to a multiple applications. Component files are compiled and can only be used by CB itself. Therefore, to convert a CB program that uses components to IWB requires one of two things:

1. The User has the original CB source file hat was used to create the component, or
2. The User completely rewrite all the subroutines contained in the component.

The following assumes the User has the original source file:

In CB, the following is the source code for a valid component.

```

InitMySubs
  DECLARE mysub1(x: INT, y: INT)
  DECLARE mysub2(x: STRING, y: STRING)
RETURN

SUB mysub1(x, y)
  z: INT
  z=x + y
RETURN z

SUB mysub2(x, y)
  z: STRING
  z=x + y
RETURN z

```

and the following is an example of the calling program

```

GOSUB InitMySubs
...
a=1
b=2
PRINT mysub1(a, b)

c = "My dog "
d = "has fleas"

```

```
PRINT mysub2(c, d)
```

In IWB, the User has two options:

One is to simply add the subroutines to the same main file the User has been converting to. In that case the code would look like this:

```
a=1
b=2
PRINT mysub1(a, b)

c = "My dog "
d = "has fleas"
PRINT mysub2(c, d)

SUB mysub1(x: INT, y: INT), INT
  z: INT
  z=x + y
RETURN z
ENDSUB

SUB mysub2(x: STRING, y: STRING), STRING
  z: STRING
  z=x + y
RETURN z
ENDSUB
```

But taking this option is not in the original spirit on the component code, which was to make the code easily reusable in multiple applications. This leads to the second option the user has.

The User creates a second source file to contain the converted component code which will now look like this:

```
GLOBAL SUB mysub1(x: INT, y: INT), INT
  z: INT
  z=x + y
RETURN z
ENDSUB

GLOBAL SUB mysub2(x: STRING, y: STRING), STRING
  z: STRING
  z=x + y
RETURN z
ENDSUB
```

Up until the introduction of a second IWB source the User was dealing with a single file application and could follow the procedures outlined in the [How-To»Single File Applications](#) section. With the introduction of a second IWB source file the User will be required to create a project following the procedures outlined in the [How-To»Projects](#) section.

The calling IWB program(that the CB source file is being converted to will appear like this

```
$MAIN
DECLARE EXTERN, mysub1(x: INT, y: INT), INT
DECLARE EXTERN,mysub2(x: STRING, y: STRING), STRING
a=1
b=2
PRINT mysub1(a, b)

c = "My dog "
d = "has fleas"
```

```
PRINT mysub2(c, d)
```

The above file and the converted component file are then both added to the project.

NOTE: If additional IWB source files are added to the project the DECLARE EXTERN statements will have to be added at the top of any file that calls a component subroutine. Sometimes this is best done by using an \$INCLUDE file.

Subroutines - DLL Calls

There are many times when functions are called that reside in external *.dll files. Before the functions can be used they have to be declared. In general, there are four types of declarations.

In CB, the most common is

```
DECLARE "User32", MessageBoxA (wnd:window, text:string, title:string, flags:int), int
```

In IWB it becomes

```
DECLARE IMPORT, MessageBoxA (wnd:window, text:string, title:string, flags:int), int
```

In some cases an alias is required for a function to resolve naming conflicts.

In CB it appears as

```
DECLARE "User32", MBox ALIAS MessageBoxA (wnd:window, text:string, title:string, flags:int), int
```

and in IWB it becomes

```
DECLARE IMPORT, MBox ALIAS MessageBoxA (wnd:window, text:string, title:string, flags:int), int
```

Some functions utilize a C calling convention.

In CB, this is signified by an ! in front of the dll name like

```
DECLARE "!user32.dll", wsprintf(lpstr:STRING, lpctr:STRING, OptionalArguments as POINTER), INT
```

In IWB it becomes

```
DECLARE CDECL IMPORT, wsprintfA(buf as STRING, format as STRING, ...), INT
```

There is a special case when using functions in the C Runtime Library

In CB, the declaration looks like the same as in the C calling convention above

```
DECLARE "!crt.dll", sprintf(out as string, format as string, value as double), INT
```

In IWB it is as follows

```
DECLARE CDECL EXTERN _sprintf(buf as STRING, format as STRING, ...), INT
```

Notice the function name is preceded by an underscore which is usually the case for the *C Runtime Library* functions.

In all four examples above, when converting to IWB, the declarations no longer contain a dll name. The next logical question is how does the compiler know where to look for the functions when building the application?

IWB requires for each dll file where the User calls a function that an import library exist. The details of creating an import library is covered in the [Utilities / Create Import Library](#) section. The resulting *.LIB file is saved in the \LIBS folder.

The User then adds a statement, similar to the following, at the beginning of the IWB file.

IWBASIC comes with the following import libraries in the \LIBS folder:

```
kernel32.lib
user32.lib
gdi32.lib
comdlg32.lib
comctl32.lib
shell32.lib
winmm.lib
ole32.lib
olepro32.lib
oleaut32.lib
winspool.lib
shlwapi.lib
uuid.lib
ddraw.lib
dinput.lib
advapi32.lib
crtDll.lib
ddraw.lib
dinput.lib
dsound.lib
odbc32.lib
rasapi32.lib
ws2_32.lib
wsock32.lib
```

Windows / Dialogs

The following changes are required only if the source file creates a window or dialog.

In CB, the following creates a window

```
WINDOW w,0,0,700,500,style,0,"A CB Window", msghandler
```

In IWB, it becomes

```
OPENWINDOW w,0,0,700,500,style,0,"A CB Window", &msghandler
```

Notice the '&' symbol preceding the name of the message handling subroutine.

In CB, the message handling subroutine looks like

```
SUB msghandler
...
RETURN
```

In IWB, it appears as

```
SUB msghandler( ) , INT
...
RETURN 0
ENDSUB
```

IWB message handlers always return a FALSE except in a few advanced programming cases.

Dialogs follow the same pattern as windows.

In CB, the following creates a dialog

```
DIALOG w,0,0,700,500,style,0,"A CB Window", msghandler
```

In IWB, it becomes

```
CREATEDIALOG w,0,0,700,500,style,0,"A CB Window", &msghandler
```

Notice the '&' symbol preceding the name of the message handling subroutine.

In CB, the message handling subroutine looks like

```
SUB msghandler
...
RETURN
```

In IWB, it appears as

```
SUB msghandler( ) , INT
...
RETURN 0
ENDSUB
```

IWB message handlers always return a FALSE except in a few advanced programming cases.

Controls

All Controls follow the same pattern.

In CB, a control is created with

```
CONTROL win, "B, Exit, 500, 320, 70, 30, @ctlbtnflat, 7"
```

In IWB the same control would be:

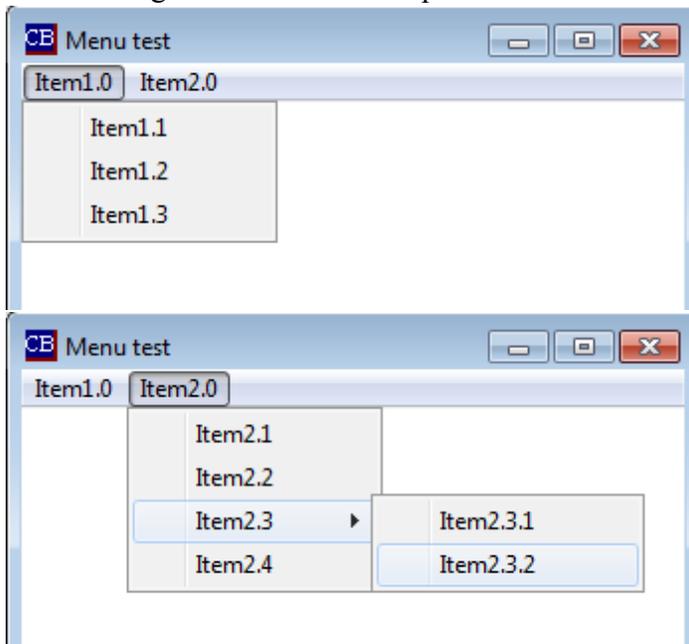
```
CONTROL win, @BUTTON, "Exit", 500, 320, 70, 30, @ctlbtnflat, 7
```

In both cases the first parameter after the window/dialog variable identifies the type of control. The following shows the CB to IWB conversion values.

B - @BUTTON
 E - @EDIT
 S - @SCROLLBAR
 R - @RADIOBUTTON
 C - @CHECKBOX
 L - @LISTBOX
 M - @COMBOBOX
 T - @STATIC
 LV - @LISTVIEW
 RE - @RICHEDIT
 SW - @STATUS

Menus

The following discussion is based upon the menu shown in the following two images



In CB, either of the following code blocks will create the above *Item1.0* menu and its sub-items:

```
MENU w1,"T,Item1.0,0,0", "I,Item1.1,0,1","I,Item1.2,0,2","I,Item1.3,0,3"
```

OR

```
f1$ = "T,Item1.0,0,0"
f2$ = "I,Item1.1,0,1"
f3$ = "I,Item1.2,0,2"
f4$ = "I,Item1.3,0,3"
MENU w1,f1$, f2$, f3$, f4$
```

It should be noted that the entire menu could have been created by expanding this code to cover all the entries.

In IWB, the same menu entries are created with

```
BEGINMENU w1
  MENUTITLE "Item1.0"
  MENUITEM "Item1.1,0", 0, 1
  MENUITEM "Item1.2,0", 0, 2
  MENUITEM "Item1.3,0", 0, 3
ENDMENU
```

In CB, the rest of the menu is completed with one of the two following code blocks.

```
INSERTMENU w1,1, "T,Item2.0,0,0", "I,Item2.1,0,4", "I,Item2.2,0,5", "S,Item2.3,0,0", "I,Item2.3.1,
```

or

```
m1$ = "T,Item2.0,0,0"
m2$ = "I,Item2.1,0,4"
m3$ = "I,Item2.2,0,5"
m4$ = "S,Item2.3,0,0"
m5$ = "I,Item2.3.1,0,6"
m6$ = "I,Item2.3.2,0,7"
m7$ = "^I,Item2.4,0,8"
INSERTMENU w1,1, m1$, m2$, m3$, m4$, m5$, m6$, m7$
```

Of special note is the contents of m4\$ which is the start of a popup menu and m7\$ where the "^" indicates to move back one level to the left.

In IBW, the corresponding code is

```
BEGININSERTMENU w1, 1
  MENUTITLE "Item2.0"
  MENUITEM "Item2.1,0", 0, 4
  MENUITEM "Item2.2,0", 0, 5
  BEGINPOPOPUP "Item2.3,0"
    MENUITEM "Item2.3,1", 0, 6
    MENUITEM "Item2.3,2", 0, 7
  ENDPPOPUP
  MENUITEM "Item2.4,0", 0, 8
ENDMENU
```

Menus - Context

Context menus follow the same general structure as for the menus described above.

In CB, a context menu is created with

```
CONTEXTMENU w, @MOUSEX, @MOUSEY, "I,Color,0,99","I,Clear,0,1"
```

In IWB, it is coded as follows

```
CONTEXTMENU w,@MOUSEX,@MOUSEY
  MENUITEM "Color",0,99
  MENUITEM "Clear",0,1
ENDMENU
```

INSTR

The INSTR command has a different order for passed parameters.

In CB, the syntax is

```
Position = INSTR({start, } string1, string2)
```

In IWB, the syntax changed to:

```
Position = INSTR(string1, string2, {start})
```

ListView

There is a difference when reading the text in a ListView.

In CB, the syntax is

```
a$ = CONTROLCMD(d, 10, @LVGETTEXT, lv.iItem, item)
```

In IWB, the syntax is

```
CONTROLCMD(d, 10, @LVGETTEXT, lv.iItem ,item, a$)
```

2D / 3D Graphics

When converting 2D or 3D graphics there is very little that can be ported.

While there are a few commands with the same name, the vast majority of the commands are completely different.

It will be necessary to re-write most of the graphics using the different IWB graphics commands.

IWB does not use the @IDDXUPDATE message handler as does CB. This mean the CASE @IDDXUPDATE statement and any associated code will have to be removed from the IWB version.

Conclusion

IWB is more stringent in its use of variable typing, so the User may find other error or warning messages occurring for a particular program.

Hopefully, making the changes listed above, the converted program will spring to life in IWBasic.

11.8 Convert from EBasic / IBPro

EBasic requires only two changes to get rid of nuisance warnings when compiling in IWBasic.

In EBasic, the message handling subroutine looks like

```
SUB msghandler
...
RETURN
ENDSUB
```

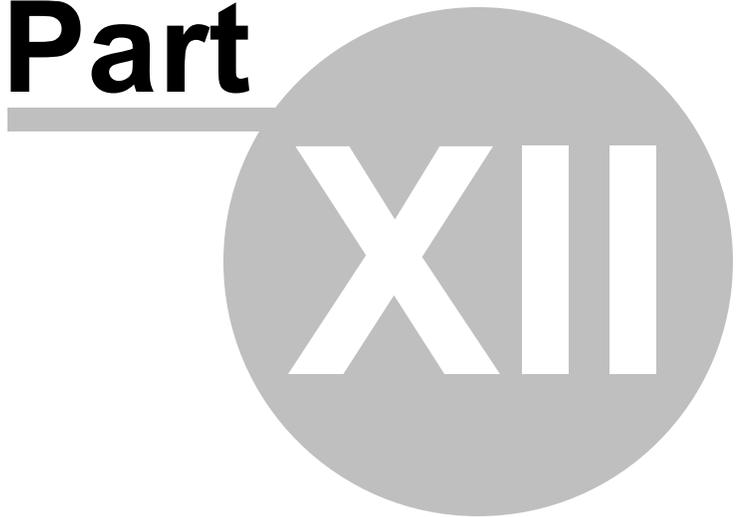
In IWBasic, it appears as

```
SUB msghandler( ) , INT
...
RETURN 0
ENDSUB
```

IWB message handlers always return a FALSE except in a few advanced programming cases.

Alphabetical Command Reference

Part



12 Alphabetical Command Reference

12.1 ABS

Syntax

DOUBLE = ABS(num as DOUBLE)

Description

Returns the absolute value of a number.

Parameters

num - number to take the absolute value of.

Return value

The absolute value is returned as a double precision number.

Remarks

None

Example usage

```
PRINT ABS (-1.1)
```

12.2 ADDACCELERATOR

Syntax

ADDACCELERATOR(win as WINDOW, fVirt as CHAR, key as WORD, cmd as WORD)

Description

Adds an accelerator (shortcut key) to the window or dialog.

Parameters

win - Window or Dialog to add the accelerator to.

fVirt - Flag indicating which combination of keys activates the accelerator.

key - ASCII or virtual key code.

cmd - ID of menu associated with the accelerator.

Return value

None

Remarks

fVirt can be an or'ed combination of:

@FCONTROL - CTRL key must be held down with the specified key.

@FALT - ALT key must be held down with the specified key.

@FSHIFT - SHIFT key must be held down with the specified key.

@FNOINVERT - Specifies that no top-level menu item is highlighted when the accelerator is used. If this flag is not specified, a top-level menu item will be highlighted, if possible, when the accelerator is used.

@FVIRTKEY - *key* specifies a virtual key code. If this flag is not used then *key* specifies an ASCII key code. Virtual key codes are listed in the appendix of this users guide.

Example usage

```
BEGINMENU win
  MENUTITLE "&File"
  MENUITEM "&Load File\tCtrl+L",0,1
  MENUITEM "&Save\tCtrl+S",0,2
  MENUITEM "&Print\tAlt+P",0,4
  MENUITEM "&Quit\tCtrl+C",0,3
  MENUTITLE "&Options"
  MENUITEM "Change Font\tF4",0,5
ENDMENU

'add our keyboard accelerators
ADDACCELERATOR win,@FCONTROL|@FVIRTKEY,ASC("L"),1
ADDACCELERATOR win,@FCONTROL|@FVIRTKEY,ASC("S"),2
ADDACCELERATOR win,@FCONTROL|@FVIRTKEY,ASC("C"),3
ADDACCELERATOR win,@FALT|@FVIRTKEY,ASC("P"),4
ADDACCELERATOR win,@FVIRTKEY,0x73,5:'F4 key changes font
```

12.3 ACOS

Syntax

DOUBLE = ACOS(num as DOUBLE)

Description

Calculates the Arccosine of a number.

Parameters

num - The number to take the Arccosine

Return value

The angle in radians.

Remarks

See also [ACOSD](#), [FACOS](#), [FACOSD](#)

Example usage

```
angle = ACOS(COS(.866025))
```

12.4 ACOSD

Syntax

DOUBLE = ACOSD(num as DOUBLE)

Description

Calculates the Arccosine of a number.

Parameters

num - The number to calculate the arccosine of.

Return value

The angle in degrees.

Remarks

See also [ACOS](#), [FACOS](#), [FACOSD](#)

Example usage

```
PRINT ACOSD(COSD(60))
```

12.5 ADDMENUITEM

Syntax

ADDMENUITEM(*win* as WINDOW,*pos* as UINT,*text* as STRING,*flags* as UINT,*id* as UINT)

Description

Adds an item to an already existing menu by position.

Parameters

win - WINDOW or DIALOG containing menu.

pos - Zero based position of the popup menu to add item to.

text - The text of the new menu item.

flags - State flag of the new menu item.

id - The new menu item identifier.

Return value

None

Remarks

New item is added to the end of the existing popup menu.

See Also: [REMOVEMENUITEM](#), [BEGINMENU](#), [BEGININSERTMENU](#)

Example usage

```
ADDMENUITEM mainwnd, 0, "Quit", 0, 75
```

12.6 ADDSTRING

Syntax

ADDSTRING(*win* as WINDOW,*id* as UINT,*str* as STRING)

Description

Adds a string to a list box or combo box control

Parameters

win - WINDOW or DIALOG containing the control

id - Identifier of the control

str - String to add to the control

Return value

None

Remarks

String is added to the end of the list unless sorting is specified in the style of the control.

See Also: [INSERTSTRING](#), [DELETESTRING](#)

Example usage

```
ADDSTRING mydlg, 10, "New text"
```

12.7 ALIAS

Syntax

ALIAS

Description

Reserved word. Used with the DECLARE statement.

Parameters

None

Return value

None

Remarks

See Also: [DECLARE](#)

Example usage

```
DECLARE IMPORT, ZeroMemory ALIAS RtlZeroMemory(pvoid as POINTER, length as INT), INT
```

12.8 AllocHeap

Syntax

UINT = AllocHeap(nSize as UINT)

Description

Used internally by string functions. Allocates memory from the programs heap.

Parameters

nSize - Number of bytes to allocate

Return value

A pointer stored as a UINT to the heap memory.

Remarks

This function should only be used by command implementers.

See Also: [FreeHeap](#)

Example usage

```
DEF p as POINTER
p = AllocHeap(100)
#<STRING>p = "Copied to heap memory"
PRINT #<STRING>p
FreeHeap(p)
```

12.9 ALLOCMEM

Syntax

INT = ALLOCMEM(mem as MEMORY, count as INT, size as INT)

Description

Allocates memory for use within your program. Memory must be freed with the FREEMEM command when your program is finished using it. Total memory size will be equal to count * size. All bytes of the block of memory are initialized to zero.

Parameters

mem - A variable of type MEMORY used as a placeholder to access the allocated memory

count - The number of elements to allocate

size - The size of each element

Return value

Returns 0 on success or -1 on failure. Function may fail if the size requested is larger than the amount of free memory available.

Remarks

See also [FREEMEM](#), [READMEM](#), [WRITEMEM](#)

Example usage

```
DEF mymem as MEMORY
IF ALLOCMEM(mymem,100,10)
    PRINT "Allocated 1000 bytes of memory"
    FREEMEM mymem
ENDIF
```

12.10 APPEND\$

Syntax

STRING = APPEND\$(str1 as STRING, ...)

Description

Concatenates all of the strings in the parameter list and returns the total string.

Parameters

str1 - The first string in the parameter list

... - One or more additional strings to concatenate

Return value

A string containing all of the input strings concatenated.

Remarks

Make sure the string you assign to this function is large enough to hold the result. IW BASIC does not check for overwritten string memory. The function requires a minimum of two input strings.

See also: Operator '+'

Example usage

```
A$ = APPEND$("this ", "is ", "a ", "string")  
PRINT A$
```

12.11 APPENDMENU

Syntax

APPENDMENU(hmenu as UINT,item as STRING,flags as INT,id as UINT)

Description

Low level menu function used internally by menu creation macros. This function is equivalent to the Windows API AppendMenu call.

Parameters

hmenu - HANDLE to an existing menu or popup menu.

item - Text for new menu item.

flags - Menu creation flags.

id - Menu ID or handle to a popup menu.

Return value

None

Remarks

Used only for special cases where the menu creation macros do not provide enough functionality.

See Also: [CREATEMENU](#), [SETMENU](#)

Example usage

```
hMenu = CreateMenu()  
hPopup = CreateMenu(1)  
APPENDMENU(hMenu, "File", MF_POPUP|MF_STRING, hPopup)  
APPENDMENU(hPopup, "Open", MF_STRING, 1)  
APPENDMENU(hPopup, "Quit", MF_STRING, 2)  
SETMENU win, hMenu
```

12.12 ASC

Syntax

INT = ASC(str as STRING)

Description

Returns the ASCII value of the first character in the input string.

Parameters

str - A string parameter. Only the first character is converted

Return value

The ASCII value of the character

Remarks

See Also: [CHRS](#)

Example usage

```
PRINT ASC("A")
```

12.13 ASIN

Syntax

DOUBLE = ASIN(num as DOUBLE)

Description

Calculates the arcsine of a number.

Parameters

num - The number to calculate the arcsine.

Return value

The angle in radians of the arcsine of *num*.

Remarks

See Also: [ASIND](#), [FASIN](#), [FASIND](#)

Example usage

```
PRINT ASIN(SIN(.8543))
```

12.14 ASIND

Syntax

DOUBLE = ASIND(num as DOUBLE)

Description

Calculates the arcsine of a number.

Parameters

num - The number to calculate the arcsine.

Return value

The angle in degrees of the arcsine of *num*.

Remarks

See Also: [ASIN](#), [FASIN](#), [FASIND](#)

Example usage

```
PRINT ASIND(SIND(45.0))
```

12.15 ATAN

Syntax

DOUBLE = ATAN(num as DOUBLE)

Description

Calculates the arctangent of a number.

Parameters

num - The number to calculate the arctangent.

Return value

The angle in radians of the arctangent of *num*.

Remarks

See Also: [ATAND](#), [FATAN](#), [FATAND](#)

Example usage

```
PRINT ATAN(1.113)
```

12.16 ATAND

Syntax

DOUBLE = ATAND(num as DOUBLE)

Description

Calculates the arctangent of a number.

Parameters

num - The number to calculate the arctangent..

Return value

The angle in degrees of the arctangent of *num*.

Remarks

See Also: [ATAN](#), [FATAN](#), [FATAND](#)

Example usage

```
PRINT ATAND(47.0)
```

12.17 ATTACHBROWSER

Syntax

INT = ATTACHBROWSER(win as WINDOW,OPT url as STRING)

Description

Embeds a browser control into the window specified.

Parameters

win - Window to embed the browser control into.

url - Optional. Initial URL to browse to.

Return value

Returns 0 if the browser was successfully initialized and attached to the window or -1 on failure.

Remarks

Once embedded into a window the window will contain the browser control until it is destroyed with CLOSEWINDOW. The browser is automatically sized to the client area. The window should be created with the @NOAUTODRAW style to prevent flickering when the browser control is resized.

The browser control requires Internet Explorer 4.0 or greater to be installed on the system

See Also: [BROWSECMD](#), sample browser_demo.iwb

Example usage

```
IF ATTACHBROWSER(mywin, "http://www.ionicwind.com") = -1
    MESSAGEBOX mywin, "Unable to create browser", "Error"
ENDIF
```

12.18 AUTODEFINE

Syntax

AUTODEFINE *state*

Description

Controls the auto definition of variables by assignment.

Parameters

state - "ON" or "OFF"

Return value

None

Remarks

Defaults to "ON". The compiler will automatically define a variable when its first assigned a value if it hasn't been previously defined. The type of the variable will be determined by the value assigned. If set to "OFF" all variables must be defined before use.

See Also: [DEF / DIM](#)

Example usage

```
AUTODEFINE "OFF"
```

12.19 BACKPEN

Syntax

BACKPEN(*win* as WINDOW,*bkclr* as UINT)

Description

Sets the background color for drawing operations in a window

Parameters

win - Window to change color.

bkclr - New background drawing color .

Return value

None

Remarks

The background color is used as the text fill color when text mode is set to @OPAQUE

See Also: [FRONTPEN](#)

Example usage

```
BACKPEN mywin, RGB(0,0,255)
```

12.20 BASELEN

Syntax

size = BASELEN(variable)

size = BASELEN(UDT)

Description

Returns the length in bytes of a variable or UDT.

Parameters

variable - Any defined variable.

UDT - The name of a user data type as specified in the TYPE statement.

Return value

Depends on the type of variable:

FILE, BFILE - Returns the length of the file, file must have been opened with OPENFILE.

MEMORY - Returns the size of the memory allocated with ALLOCMEM.

UDT - Returns the actual size a UDT takes in memory using the packing value specified in the TYPE statement

STRING - Returns the string length, not the defined length.

Remarks

For a MEMORY variable only memory obtained with ALLOCMEM will return a size.

Unlike the LEN command BASELEN returns the element size for arrays instead of the entire array length. BASELEN is used internally by many commands.

See Also: [ALLOCMEM](#), [DEF / DIM](#), [TYPE](#), [LEN](#)

Example usage

```
mystring = "This is a test"  
PRINT BASELEN(mystring)
```

12.21 BEGININSERTMENU

Syntax

BEGININSERTMENU win, position

Description

Inserts one or more menu(s) into an existing menu in a window or dialog.

Parameters

win - Window or dialog with menu to modify.

position - Zero based insertion position.

Return value

None.

Remarks

Commonly used for MDI frame windows. Every BEGININSERTMENU must be paired with an ENDMENU statement.

See Also: [BEGINMENU](#), [ENDMENU](#), [MENUITEM](#), [MENUTITLE](#), [BEGINPOPUP](#), [ENDPOPUP](#), [SEPARATOR](#), [BEGINPOPUP](#)

Example usage

```
BEGININSERTMENU frame,0
  MENUTITLE "&File"
  MENUITEM "&New",0,1
  MENUITEM "&Quit",0,2
ENDMENU
```

12.22 BEGINMENU

Syntax

BEGINMENU win as WINDOW

Description

Begins defining a menu for a window or dialog. The window or dialog must be open before using this macro.

Parameters

win - Window or dialog to add menu to.

Return value

None

Remarks

Every BEGINMENU statement must be paired with an ENDMENU statement. This is a high level macro for adding an unlimited number of menu items, pop-ups and titles. The menu macros are translated by the compiler into the appropriate calls to CREATEMENU, APPENDMENU, INSERTMENU and SETMENU.

See Also: [ENDMENU](#), [MENUITEM](#), [MENUTITLE](#), [BEGINPOPUP](#), [ENDPOPUP](#), [SEPARATOR](#), [BEGININSERTMENU](#), [CONTEXTMENU](#)

Example usage

```
BEGINMENU win
  MENUTITLE "&File"
  MENUITEM "Open",0,1
  MENUITEM "Close",0,2
  BEGINPOPUP "Save As..."
    MENUITEM "Ascii",0,3
    MENUITEM "Binary",0,4
  ENDPOPUP
```

```
SEPARATOR
MENUITEM "&QUIT",0,5
MENUTITLE "&Edit"
MENUITEM "Cut",0,6
ENDMENU
```

12.23 BEGINPOPUP

Syntax

BEGINPOPUP title as STRING

Description

Begins definition of a popup menu (sub menu) inside of a menu definition macro.

Parameters

title - Title of the popup menu

Return value

None

Remarks

Every BEGINPOPUP statement must be paired with an ENDPOPUP. All menu items defined between the two will be located in the popup menu. Popup menus can be nested to any level.

See Also: [BEGINMENU](#), [ENDMENU](#), [MENUITEM](#), [MENUTITLE](#), [ENDPOPUP](#), [SEPARATOR](#), [BEGININSERTMENU](#)

Example usage

```
BEGINMENU win
  MENUTITLE "&File"
  MENUITEM "Open",0,1
  MENUITEM "Close",0,2
  BEGINPOPUP "Save As..."
    MENUITEM "Ascii",0,3
    MENUITEM "Binary",0,4
  ENDPOPUP
  SEPARATOR
  MENUITEM "&QUIT",0,5
  MENUTITLE "&Edit"
  MENUITEM "Cut",0,6
ENDMENU
```

12.24 BFILE

Syntax

bfile = BFILE(num)

Description

Converts a value to a bfile.

Parameters

num - A UINT

Return value

If *num* is not a file pointer the results will be unpredictable.

Remarks**Example usage**

```
BFILE (fptr)
```

12.25 BREAK

Syntax

BREAK

Description

Terminates the execution of the nearest enclosing loop (FOR-NEXT, DO-UNTIL, WHILE-ENDWHILE)

Parameters

none

Return value

none

Remarks

BREAK allows an early exit from the nearest FOR/NEXT, DO/UNTIL, or WHILE/ENDWHILE loop before the loop has finished all of the iterations.

The command can also be used with a FOR/EACH loop where BREAKFOR cannot.

See Also: [FOR](#), [NEXT](#), [DO](#), [UNTIL](#), [WHILE](#), [ENDWHILE](#), [Loop Statements](#)

Example usage

```
FOR x = 1 TO 100
  IF x = 50 THEN BREAK
NEXT x
PRINT x
```

12.26 BREAKFOR

Syntax

BREAKFOR

Description

Exits a FOR/NEXT loop early

Parameters

none

Return value

none

Remarks

BREAKFOR allows an early exit of a FOR/NEXT loop before the loop has finished all of the iterations.

The command cannot be used with a FOR/EACH loop.

See Also: [FOR](#), [NEXT](#), [Loop Statements](#)

Example usage

```
FOR x = 1 TO 100
  IF x = 50 THEN BREAKFOR
NEXT x
PRINT x
```

12.27 BROWSECMD

Syntax

INT = BROWSECMD(win as WINDOW,cmd as INT,...)

Description

Controls the embedded browser in a window.

Parameters

win - Window containing the browser control

cmd - Command to send to browser

... - Variable list of additional parameters based on the command

Return value

Dependent on the command

Remarks

Currently available commands and their parameters:

@NAVIGATE, strURL

@GOBACK

@GOFORWARD

@GOHOME

@BROWSESTOP

@REFRESH

@BROWSESEARCH

@BROWSELOAD, strHTML
@BACKENABLED : Returns TRUE or FALSE
@FORWARDENABLED: Returns TRUE or FALSE
@GETTITLE, pStrTitle {,cchTitle}
@BROWSEPRINT
@GETNAVURL, pStrURL {,cchUrl}
@GETPOSTDATA, pStrData {,cchData}
@GETHEADERS, pStrHeaders {,cchHeaders}
@CANCELNAV
@GETSTATUSTEXT, pStrStatus {,cchStatus}

See Also: [ATTACHBROWSER](#)

Example usage

```
BROWSECMD (wnd, @GETSTATUSTEXT, caption, 255)
```

12.28 BYTE

Syntax

byte = BYTE(num)

Description

Converts a value to a byte.

Parameters

num - Any numeric

Return value

A single byte value.

Remarks

Example usage

```
print BYTE(65+256)
```

12.29 CalendarControl

Syntax

UINT = CalendarControl(win as WINDOW, l as INT, t as INT, w as INT, h as INT, flags as INT, exStyle as INT, id as UINT)

Description

Creates a calendar control, also known as a Month Calendar.

Parameters

win - Parent window or dialog containing the control.

l,t,w,h - Position and dimension of the control.

flags - Styles for the control.

exStyle - Extended window styles for the control.

id - Control identifier.

Return value

If used with a window the return value is the handle of the control. If used with a dialog the return value is 0 and the handle to the control can be obtained with GETCONTROLHANDLE during the @IDINITDIALOG message

Remarks

The control can also be created using CONTROLEX with a window class of "SysMonthCal32"

Example usage

```
CalendarControl cp,0,0,0,0,@BORDER|@TABSTOP,0,IDCALENDAR
ccGetMinimumRect cp,IDCALENDAR,rcTemp
SETSIZE cp,40,80,rcTemp.right,rcTemp.bottom,IDCALENDAR
ccSetColor cp,IDCALENDAR,MCSC_MONTHBK,RGB(202,202,202)
```

12.30 CALLOBJECTMETHOD

Syntax

INT = CallObjectMethod(IDispatch obj,STRING member,...)

Description

Calls a method of a COM object using script syntax.

Parameters

obj - The Object returned by the CREATECOMOBJECT command.

member - The name of the method to call.

... - Optional parameter list.

Return value

0 for Success.

Remarks

Parameters to a method are specified by using C printf-like specifiers. The table below shows the supported specifiers:

Identifier	Type
%d	INT
%u	UINT
%e	DOUBLE
%b	INT
%v	VARIANT UDT.

%B	BSTR - Created with the AllocSysString API function
%s	STRING
%S	WSTRING
%T	WSTRING
%o	IDispatch COM object
%O	IUnknown COM object
%t	C time_t UDT
%W	SYSTEMTIME UDT
%f	FILETIME UDT
%D	C date type.
%p	POINTER
%m	Specifies a missing/optional argument

Example usage

```
IDispatch Connection
POINTER szResponse
INT _status
Connection = CreateComObject("MSXML2.XMLHTTP.3.0", "")
CallObjectMethod(Connection, ".Open(%s, %s, %b)", "GET", "http://myserver.com/test.xml")
CallObjectMethod(Connection, ".Send")
GetComProperty(Connection, "%d", &_status, ".status")
if _status = 200
    GetComProperty(Connection, "%T", &_szResponse, ".ResponseXML.xml")
    PRINT w2s(*<wstring>_szResponse)
    FreeComString(_szResponse)
endif
Connection->Release()
```

12.31 CASE&

Syntax

CASE& value

Description

An inclusive test condition for a SELECT statement.

Parameters

value - value to compare with the SELECT parameter.

Return value

None

Remarks

Use to group test cases together. One or more CASE& statements can follow an initial CASE

See Also: [CASE](#), [SELECT](#), [DEFAULT](#), [ENDSELECT](#)

Example usage

```
SELECT a
```

```
CASE 1
CASE& 2
CASE& 3
    PRINT "Its 1, 2 or 3"
ENDSELECT
```

12.32 CASE

Syntax

CASE value

Description

A test condition for a SELECT statement.

Parameters

value - value to compare with the SELECT parameter.

Return value

None

Remarks

If the value in the select statement and the value in the CASE statement are equal then the statements immediately after this CASE will execute. CASE statements are non inclusive. To include more than one CASE for a group of statements see the CASE& statement.

See Also: [CASE&](#), [DEFAULT](#), [SELECT](#), [ENDSELECT](#)

Example usage

```
A = 1
SELECT A
    CASE 1
        PRINT "TRUE!"
    CASE 2
        PRINT "You wont see this text"
    DEFAULT
        PRINT "None of the above"
ENDSELECT
```

12.33 CATCH

Syntax

CATCH

Description

The start of a block of code that handles exceptions from a TRY block..

Parameters

None

Return value

None

Remarks

See Also: [ENDCATCH](#), [Exception Handling](#)

Example usage

```
exec_a:
  try
    print 1/a
  endtry
  catch
    a=1
    goto exec_a
  endcatch
```

12.34 cbeAddString

Syntax

cbeAddString(win as WINDOW, id as UINT, text as STRING, OPT image=-2 as INT, OPT selimage=-2 as INT)

Description

Adds a string item to a ComboBoxEx control. String is added to the end of the list.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

text - Text to be added.

image - Optional. Zero based index into the image list.

selimage - Optional. Zero based index into the image list.

Return value

None

Remarks

Equivalent to the ADDSTRING command for normal combobox controls.

Example usage

```
ComboBoxEx cp, 0, 0, 100, 100, @CTCOMBODROPDOWN|@VSCROLL, 0, IDC_COMBOBOX
cbeAddString cp, IDC_COMBOBOX, "String1"
cbeAddString cp, IDC_COMBOBOX, "String2"
cbeAddString cp, IDC_COMBOBOX, "String3"
```

12.35 cbeDeleteString

Syntax

`cbeDeleteString(win as WINDOW,id as UINT,pos as INT)`

Description

Removes a string from a ComboBoxEx control. Remaining strings are moved up to fill the empty position.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

pos - Zero based position of the string to delete.

Return value

None

Remarks

Equivalent to the DELETESTRING command for regular combobox controls.

Example usage

```
cbeDeleteString cb, IDC_COMBOBOX, 5
```

12.36 cbeGetSelected

Syntax

`INT = cbeGetSelected(win as WINDOW,id as UINT)`

Description

Returns the zero-based index of the currently selected item in a ComboBoxEx control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

The zero based index of the currently selected item.

Remarks

Equivalent to the GETSELECTED command for normal combobox controls.

Example usage

```
pos = cbeGetSelected(rb, IDC_COMBOBOX)
```

12.37 cbeGetString

Syntax

`STRING = cbeGetString(win as WINDOW,id as UINT,pos as INT)`

Description

Returns a string in a ComboBoxEx control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

pos - The zero based position of the string to retrieve.

Return value

A string.

Remarks

Equivalent to GETSTRING

Example usage

```
MESSAGEBOX rb,cbeGetString(rb,IDCOMBOBOX,cbeGetSelected(rb,IDCOMBOBOX)), "Selected"
```

12.38 cbeGetStringCount

Syntax

```
INT = cbeGetStringCount(win as WINDOW,id as INT)
```

Description

Returns the number of strings in a ComboBoxEx control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

The number of strings in the control.

Remarks

Equivalent to the GETSTRINGCOUNT command for regular combo box controls.

Example usage

```
n = cbeGetStringCount(cp, IDCOMBOBOX)
FOR x = 0 to n-1
    item = cbeGetString(cp, IDCOMBOBOX, x)
    SaveString(item)
NEXT x
```

12.39 cbInsertString

Syntax

```
cbInsertString(win as WINDOW,id as UINT,text as STRING,pos as INT,OPT image=-2 as
```

INT,OPT selimage=-2 as INT)

Description

Inserts a string into a ComboBoxEx control.

Parameters

win - Window or dialog containing the control.

id - Identifier of the control.

pos - Zero based position to insert the string

text - String to insert.

image - Optional. Zero based index into the image list.

selimage - Optional. Zero based index into the image list.

Return value

None.

Remarks

All other strings are moved down by one position. Equivalent to the INSERTSTRING command.

Example usage

```
cbeInsertString cp, IDC_COMBOBOX, "David Blaine", 19
```

12.40 cbeSetImageList

Syntax

```
cbeSetImageList(win as WINDOW, id as UINT, himl as UINT)
```

Description

Sets an image list for a ComboBoxEx control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

himl - Handle to the image list to be set for the control.

Return value

None.

Remarks

The height of images within your image list might change the size requirements of the ComboBoxEx control. It is recommended that you resize the control after sending this message to ensure that it is displayed properly.

Example usage

```
cbeSetImageList cp, IDC_COMBOBOX, myimagelist
```

12.41 cbeSetIndent

Syntax

cbeSetIndent(win as WINDOW,id as UINT,pos as INT,indent as INT)

Description

Sets the number of indent spaces to display for the item in the ComboBoxEx control. Each indentation equals 10 pixels.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

pos - Zero based position of the item to indent.

indent - Amount to indent.

Return value

None

Remarks

None

Example usage

```
FOR x = 0 to 4
    cbeSetIndent cp, IDC_COMBOBOX, x, 2
NEXT x
```

12.42 cbeSetSelected

Syntax

cbeSetSelected(win as WINDOW,id as UINT,pos as INT)

Description

Sets the currently selected item in a ComboBoxEx control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

pos - Zero based position of the item to select.

Return value

None

Remarks

Equivalent to the SETSELECTED command for normal combobox controls.

Example usage

```
cbeSetSelected cp, IDC_COMBOBOX, 2
```

12.43 ccGetColor

Syntax

UINT = ccGetColor(win as WINDOW, id as UINT, index as INT)

Description

Returns the color of an element of the calendar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Color index. See remarks.

Return value

An RGB color.

Remarks

Index can be one of the following:

MCSC_BACKGROUND - The background color (between months)

MCSC_TEXT - The dates

MCSC_TITLEBK - Background of the title

MCSC_TITLETEXT - Text color of the title.

MCSC_MONTHBK - Background within the calendar.

MCSC_TRAILINGTEXT - The text color of header & trailing days.

Example usage

```
back = ccGetColor(cp, IDCALENDAR, MCSC_MONTHBK)
```

12.44 ccGetCurSel

Syntax

ccGetCurSel(win as WINDOW, id as UINT, month as INT BYREF, day as INT BYREF, year as INT BYREF)

Description

Retrieves the currently selected date in a calendar control

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

month, day, year - Variables of type INT to receive the selected date.

Return value

None

Remarks

The command will fail with calendars created with the @MCS_MULTISELECT style.

Example usage

```
INT m,d,y  
ccGetCurSel cp, IDCALENDAR, m, d, y
```

12.45 ccGetFirstDayOfWeek

Syntax

INT = ccGetFirstDayOfWeek(win as WINDOW,id as UINT)

Description

Retrieves the first day of the week for a month calendar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

An integer value representing the first day of the week. 0 = Monday to 6 = Sunday.

Remarks

None.

Example usage

```
day = ccGetFirstDayOfWeek( cp, IDCALENDAR)
```

12.46 ccGetMinimumRect

Syntax

ccGetMinimumRect(win as WINDOW,id as UINT,rcRect as WINRECT)

Description

Retrieves the minimum size required to display a full month in a month calendar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

rcRect - A UDT of type WINRECT to receive the minimum size.

Return value

None

Remarks

The minimum required window size for a month calendar control depends on the currently selected font, control styles, system metrics, and regional settings. When an application changes anything that affects the minimum window size, or processes a WM_SETTINGCHANGE message, it should use this command to determine the new minimum size.

Note The rectangle returned by this command does not include the width of the "Today" string, if it is present. If the @MCS_NOTODAY style is not set, your application should also retrieve the rectangle that defines the "Today" string width by sending a MCM_GETMAXTODAYWIDTH message. Use the larger of the two rectangles to ensure that the "Today" string is not clipped.

Example usage

```
CalendarControl cp,0,0,0,0,@BORDER|@TABSTOP,0,IDCALENDAR
ccGetMinimumRect cp,IDCALENDAR,rcTemp
SETSIZE cp,40,80,rcTemp.right,rcTemp.bottom,IDCALENDAR
```

12.47 ccGetScrollDelta

Syntax

INT = ccGetScrollDelta(win as WINDOW,id as UINT)

Description

Retrieves the scroll rate for a month calendar control. The scroll rate is the number of months that the control moves its display when the user clicks a scroll button.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

Returns an INT value that represents the month calendar's current scroll rate.

Remarks

None.

Example usage

```
rate = ccGetScrollDelta(cp, IDCALENDAR)
```

12.48 ccGetToday

Syntax

ccGetToday(win as WINDOW,id as UINT,month as INT BYREF,day as INT BYREF,year as INT BYREF)

Description

Retrieves the date information for the date specified as "today" for a month calendar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

month, day, year - Variables of type INT to receive the current date.

Return value

None.

Remarks

"today" is either the current system date or the date set with ccSetToday.

Example usage

```
INT m, d, y
ccGetToday cp, IDCALENDAR, m, d, y
```

12.49 ccSetColor

Syntax

ccSetColor(win as WINDOW, id as UINT, index as INT, clr as UINT)

Description

Sets the color of an element of the calendar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Color index. See remarks.

clr - Specified RGB color.

Return value

None

Remarks

Index can be one of the following:

MCSC_BACKGROUND - The background color (between months)

MCSC_TEXT - The dates

MCSC_TITLEBK - Background of the title

MCSC_TITLETEXT - Text color of the title.

MCSC_MONTHBK - Background within the calendar.

MCSC_TRAILINGTEXT - The text color of header & trailing days.

Example usage

```
ccSetColor cp, IDCALENDAR, MCSC_MONTHBK, RGB(202, 202, 202)
```

12.50 ccSetCurSel

Syntax

ccSetCurSel(win as WINDOW,id as UINT,month as INT,day as INT,year as INT)

Description

Sets the currently selected date for a month calendar control. If the specified date is not in view, the control updates the display to bring it into view.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

month, day, year - The date to select.

Return value

None.

Remarks

The command will fail with calendars created with the @MCS_MULTISELECT style.

Example usage

```
ccSetCurSel cp, IDCALENDAR, 8, 25, 2008
```

12.51 ccSetFirstDayOfWeek

Syntax

ccSetFirstDayOfWeek(win as WINDOW,id as UINT,day as INT)

Description

Sets the first day of the week for a month calendar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

day - An integer value representing the first day of the week. 0 = Monday to 6 = Sunday.

Return value

None

Remarks

None

Example usage

```
ccSetFirstDayOfWeek cp, IDCALENDAR, 6
```

12.52 ccSetScrollDelta

Syntax

ccSetScrollDelta(win as WINDOW,id as UINT,delta as INT)

Description

Sets the scroll rate for a month calendar control. The scroll rate is the number of months that the control moves its display when the user clicks a scroll button.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

delta - The scrolling rate.

Return value

None.

Remarks

None.

Example usage

```
ccSetScrollDelta cp, IDCALENDAR, 2
```

12.53 ccSetToday

Syntax

ccSetToday(win as WINDOW,id as UINT,month as INT,day as INT,year as INT)

Description

Sets the "today" selection for a month calendar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

month, day, year - Date to use as the "today" selection.

Return value

None.

Remarks

If the "today" selection is set to any date other than the default, the following conditions apply:

--The control will not automatically update the "today" selection when the time passes midnight for the current day.

--The control will not automatically update its display based on locale changes.

Example usage

```
ccSetToday cp, IDCALENDAR, 10,10,2005
```

12.54 CEIL

Syntax

```
DOUBLE = CEIL(num as DOUBLE)
```

Description

The CEIL function returns the smallest integer that is greater than or equal to the input parameter

Parameters

num - Number to test

Return value

A double value containing the smallest integer

Remarks

See Also: [FLOOR](#)

Example usage

```
PRINT CEIL(2.8), CEIL(-2.8)
```

12.55 CENTERWINDOW

Syntax

```
CENTERWINDOW(win as WINDOW)
```

Description

Centers the window or dialog to the screen.

Parameters

win - A WINDOW or DIALOG variable

Return value

None

Remarks

None

Example usage

```
CENTERWINDOW win
```

12.56 CHAR

Syntax

char = CHAR(exp)

Description

Converts an expression to a single char.

Parameters

exp - Any numeric expression.

Return value

A single char value (0-255 inclusive).

Remarks**Example usage**

```
print CHAR(65+256)
'prints 65
print chr$(CHAR(65+256))
'prints A
```

12.57 CHECKMENUITEM

Syntax

CHECKMENUITEM(win as WINDOW,id as UINT,bChecked as INT)

Description

Sets or resets the checkmark next to a menu item

Parameters

win - Window or dialog containing the menu.

id - Identifier of the menu item.

bChecked - State of the checkmark.

Return value

None

Remarks

Menu is checked if *bChecked* = 1 or unchecked if *bChecked* = 0

Example usage

```
CHECKMENUITEM mywin, 99, 1
```

12.58 CHR\$ / WCHR\$

Syntax

STRING = CHR\$(num as INT)

WSTRING = WCHR\$(num as WORD)

Description

Converts an ASCII value into a character string.

Parameters

num - The ASCII value to convert to a character.

Return value

A string or Unicode string containing the single character.

Remarks

See Also: [ASC](#)

Example usage

```
PRINT CHR$(65)
PRINT W2S(WCHR$(65))
```

12.59 CIRCLE

Syntax

CIRCLE(win as WINDOW, x as INT, y as INT, r as INT, OPT outline as UINT, OPT fill as UINT)

Description

Draws a circle in the window at coordinates x,y with radius r. If a fill color is specified then the circle will be filled with that color. If an outline color is specified then the circle is outlined by that color. If neither color is specified then the circle is not filled and drawn with the current pen color specified by the FRONT PEN statement

Parameters

win - Window to draw circle into.

x - Coordinate

y - Coordinate

r - Radius

outline - Optional outline color.

fill - Optional fill color.

Return value

None

Remarks

See Also: [FRONT PEN](#), [ELLIPSE](#)

Example usage

```
CIRCLE mywin, 100,100,10, RGB(255,0,0), RGB(0,255,0)
```

12.60 CLOSECONSOLE

Syntax

CLOSECONSOLE

Description

Closes the text console window. If the text console window is not open then this command does nothing.

Parameters

None

Return value

None

Remarks

See Also: [OPENCONSOLE](#)

Example usage

```
OPENCONSOLE  
PRINT "hello"  
CLOSECONSOLE
```

12.61 CLOSEDIALOG

Syntax

CLOSEDIALOG(*dlg* as DIALOG, *OPT ret* as INT)

Description

Closes a dialog previously opened with the DOMODAL or SHOWDIALOG commands.

Parameters

dlg - The dialog to close

ret - Optional return value to send to the DOMODAL command

Return value

None

Remarks

The return parameter is ignored for dialogs shown with SHOWDIALOG. For dialogs shown with DOMODAL the *ret* parameter is traditionally set to @IDOK or @IDCANCEL to indicate the users selection but can be any value you wish.

See Also: [DOMODAL](#), [SHOWDIALOG](#)

Example usage

```
CLOSEDIALOG mydlg, @IDOK
```

12.62 CLOSEFILE

Syntax

CLOSEFILE(file)

Description

Closes an open file

Parameters

file - A FILE or BFILE variable successfully initialized with the OPENFILE command.

Return value

None

Remarks

See Also: [OPENFILE](#), [READ](#), [WRITE](#)

Example usage

```
CLOSEFILE filetext
```

12.63 CLOSEPRINTER

Syntax

CLOSEPRINTER(handle as UINT)

Description

Closes an open printer. The last page is ended and the document closed.

Parameters

handle - Handle returned by the OPENPRINTER command

Return value

None

Remarks

See Also: [OPENPRINTER](#), [WRITEPRINTER](#), [ENDPAGE](#)

Example usage

```
hPrt = OPENPRINTER(name, "Test Document", "TEXT")
IF (hPrt)
    data = "This is a test of printing"
    data = data + chr$(13)
    data = data + "This is line 2"
    WRITEPRINTER hPrt, data
    CLOSEPRINTER hPrt
ENDIF
```

12.64 CLOSEWINDOW

Syntax

CLOSEWINDOW(win as WINDOW)

Description

Closes the window previously opened with the OPENWINDOW statement.

Parameters

win - Window to close

Return value

None

Remarks

See Also: [OPENWINDOW](#)

Example usage

```
CLOSEWINDOW mywnd
```

12.65 CLS

Syntax

CLS

Description

Clears the text console window and places the caret at the first line, first character position.

Parameters

NONE

Return value

NONE

Remarks

See Also: [OPENCONSOLE](#)

Example usage

```
CLS
```

12.66 COLOR

Syntax

COLOR fg as INT, bg as INT

Description

Changes the text color of a console window.

Parameters

fg - The text color. An integer value from 0 to 15

bg - The fill color for the text. An integer value from 0 to 15

Return value

None

Remarks

The text console must have been opened with OPENCONSOLE before this command is used.

Color table:

COLOR number	Color Produced
0	BLACK
1	BLUE
2	GREEN
3	CYAN
4	RED
5	MAGENTA
6	BROWN
7	WHITE
8	GRAY
9	LIGHT BLUE
10	LIGHT GREEN
11	LIGHT CYAN
12	LIGHT RED
13	LIGHT MAGENTA
14	YELLOW
15	HIGH INTENSITY WHITE

Example usage

```
OPENCONSOLE
COLOR 9,0
PRINT "Light blue text on black background"
```

12.67 COLORREQUEST

Syntax

UINT = COLORREQUEST(win as WINDOW,OPT initcolor as UINT)

Description

Opens the standard palette dialog and returns the selected color

Parameters

win - Window or dialog to use as the parent.

initcolor - Color to show as initially selected.

Return value

RGB value of the select color

Remarks

See Also: [RGB](#)

Example usage

```
mycolor = COLORREQUEST(mywnd, RGB(0,0,0))
```

12.68 ComboBoxEx

Syntax

UINT = ComboBoxEx(win as WINDOW,l as INT,t as INT,w as INT,h as INT,flags as INT,exStyle as INT,id as UINT)

Description

Creates a ComboBoxEx control.

Parameters

win - Parent window or dialog containing the control.

l,t,w,h - Position and dimension of the control.

flags - Styles for the control.

exStyle - Extended window styles for the control.

id - Control identifier.

Return value

If used with a window the return value is the handle of the control. If used with a dialog the return value is 0 and the handle to the control can be obtained with GETCONTROLHANDLE during the @IDINITDIALOG message

Remarks

The control can also be created using CONTROLEX with a window class of "ComboBoxEx32".

Example usage

```
ComboBoxEx cp,0,0,100,100,@CTCOMBODROPDOWN|@VSCROLL,0,IDCOMBOBOX  
cbeAddString cp,IDCOMBOBOX,"String1"  
cbeAddString cp,IDCOMBOBOX,"String2"  
cbeAddString cp,IDCOMBOBOX,"String3"
```

12.69 COMENUMBEGIN

Syntax

COMREF = ComEnumBegin(IDispatch obj, STRING member, ...)

Description

Enumerates a COM collection.

Parameters

obj - The Object returned by the CREATECOMOBJECT command.

member - The name of the collection to enumerate.

... - Optional parameter list.

Return value

An enumerator interface object.

Remarks

The return value is an IDispatch object that is passed to the ComEnumNext command.

Example usage

```
IDispatch pEnum, pChild
pEnum = ComEnumBegin(xmlDoc, ".documentElement.childNodes")
do
    pChild = ComEnumNext(pEnum)
    if pChild <> NULL
        GetComProperty(pChild, "%s", &pTemp, ".childNodes.item(%d).text",0)
        #_pData.author = #<string>pTemp:FreeComString(_pTemp)
        GetComProperty(pChild, "%s", &pTemp, ".childNodes.item(%d).text",1)
        #_pData.title = #<string>pTemp:FreeComString(_pTemp)
        GetComProperty(pChild, "%s", &pTemp, ".childNodes.item(%d).text",2)
        #_pData.publisher = #<string>pTemp:FreeComString(_pTemp)
        GetComProperty(pChild, "%s", &pTemp, ".childNodes.item(%d).text",3)
        #_pData.date = #<string>pTemp:FreeComString(_pTemp)
        pChild->Release()
    endif
until pChild = NULL
pEnum->Release()
```

12.70 COMENUMNEXT

Syntax

COMREF = ComEnumNext(IDispatch obj)

Description

Enumerates a COM collection.

Parameters

obj - The Object returned by the COMENUMBEGIN command.

Return value

The next child object.

Remarks

The return value is an IDispatch object.

Example usage

```
IDispatch pEnum, pChild
pEnum = ComEnumBegin(xmlDoc, ".documentElement.childNodes")
do
    pChild = ComEnumNext(pEnum)
    if pChild <> NULL
        GetComProperty(pChild, "%s", &_pTemp, ".childNodes.item(%d).text",0)
        #_pData.author = #<string>_pTemp:FreeComString(_pTemp)
        GetComProperty(pChild, "%s", &_pTemp, ".childNodes.item(%d).text",1)
        #_pData.title = #<string>_pTemp:FreeComString(_pTemp)
        GetComProperty(pChild, "%s", &_pTemp, ".childNodes.item(%d).text",2)
        #_pData.publisher = #<string>_pTemp:FreeComString(_pTemp)
        GetComProperty(pChild, "%s", &_pTemp, ".childNodes.item(%d).text",3)
        #_pData.date = #<string>_pTemp:FreeComString(_pTemp)
        pChild->Release()
    endif
until pChild = NULL
pEnum->Release()
```

12.71 COMREF

Syntax

comref= COMREF(num)

Description

Converts a value to a comref.

Parameters

num - A UINT

Return value

If num is not a comref pointer the results will be unpredictable.

Remarks**Example usage**

```
COMREF(ptr)
```

12.72 CONST

Syntax

CONST name = value

Description

Defines a constant

Parameters

name - Name of constant

value - Integer value of constant, resolves to a UINT

Return value

None

Remarks

Constants work by direct substitution at compile time. Simple math is allowed to calculate the value and the calculations can contain the names of other constants. A constant cannot reference variables in your program as they are compiler calculated.

Operators allowed for constant calculations: =, ==, <>, !=, +, -, *, /, %, |, OR, || (xor), &, &&, AND, <<, >>, !, ~, ^, >, >=, <, <=.

See Also: [SETID](#)

Example usage

```
CONST BASE_VALUE = 0x4000
CONST SOMETHING = BASE_VALUE +1
CONST SOMETHING_ELSE = SOMETHING+1
a = SOMETHING

SELECT a
  CASE SOMETHING
    PRINT "It was something"
  CASE SOMETHING_ELSE
    PRINT "It was something else"
ENDSELECT
```

12.73 CONTEXTMENU

Syntax

CONTEXTMENU win as WINDOW, xPos as INT, yPos as INT

Description

Creates and shows a right-click context menu.

Parameters

win - Window or dialog to show menu in.

xPos, *yPos* - Upper left coordinate of context menu. Normally the mouse position.

Return value

None

Remarks

The items in the context menu use the same format as the BEGINMENU macro. The CONTEXTMENU statement must be paired with an ENDMENU statement to mark the end of the context menu.

See Also: [BEGINMENU](#), [ENDMENU](#), [MENUITEM](#), [MENUTITLE](#), [BEGINPOPUP](#), [ENDPOPUP](#), [SEPARATOR](#), [BEGININSERTMENU](#)

Example usage

```
CONTEXTMENU mywin,@MOUSEX,@MOUSEY
  MENUITEM "Color",0,99
  MENUITEM "Clear",0,1
  BEGINPOPOP "Line Size"
    MENUITEM "1", (linesize = 1) * @MENUCHECK,2
    MENUITEM "2", (linesize = 2) * @MENUCHECK,3
    MENUITEM "3", (linesize = 3) * @MENUCHECK,4
    MENUITEM "4", (linesize = 4) * @MENUCHECK,5
  ENDPOPUP
ENDMENU
```

12.74 CONTROL

Syntax

CONTROL(parent, type as UINT,title as STRING,l as INT,t as INT,w as INT,h as INT,flags as INT,id as UINT)

Description

Creates a control and either shows it in the window or adds it to the dialog template.

Parameters

parent - The window or dialog to place the control into

type - The type of the control.

title - The text of the control.

l, *t*, *w*, *h* - Control coordinates and dimensions.

flags - Style flags.

id - Identifier for the control.

Return value

None

Remarks

Available control types:

@BUTTON

@CHECKBOX

@RADIOBUTTON

@EDIT

@LISTBOX

@COMBOBOX

@STATIC

@SCROLLBAR

@GROUPBOX

@RICEDIT

@LISTVIEW

@STATUS

@SYSBUTTON
 @RGNBUTTON
 @TREEVIEW

See Also: [CREATEDIALOG](#), [OPENWINDOW](#)

Example usage

```
CONTROL win,@button,"QUIT",4,100,50,20,@TABSTOP|@CTLBTNDEFAULT,23
```

12.75 CONTROLCMD

Syntax

INT = CONTROLCMD(win as WINDOW,id as INT,cmd as INT, ...)

Description

Sends a command to a control.

Parameters

win - Window or dialog containing the control.

id - Identifier of the control.

cmd - Command to send.

... - Variable list of additional parameters based on the command.

Return value

Dependent on the command.

Remarks

See the sections on individual controls for details on usage.

Example usage

```
CONTROLCMD dl,1,@RTSETSELCOLOR,textclr
```

12.76 CONTROLEX

Syntax

CONTROLEX(parent, class as STRING,title as STRING,l as INT,t as INT,w as INT,h as INT,
 style as INT,exStyle as INT,id as UINT)

Description

Creates a control and either shows it in the window or adds it to the dialog template.

Parameters

parent - The window or dialog to place the control into

class - The class name of the control.

title - The text of the control.

l, t, w, h - Control coordinates and dimensions.

style - Style flags.
ex.Style - Extended style flags.
id - Identifier for the control.

Return value

If used with a window the return value is the handle of the control. If used with a dialog the return value is 0 and the handle to the control can be obtained with [GETCONTROLHANDLE](#) during the @IDINITDIALOG message

Remarks

See Also: [CREATEDIALOG](#), [OPENWINDOW](#), [Creating Controls](#)

Example usage

```
CONTROLEX d1,"msctls_progress32","",4,40,236,20,@BORDER,@EXCLIENEDGE,20
```

12.77 CONTROLEXISTS

Syntax

INT = CONTROLEXISTS(win as WINDOW,id as UINT)

Description

Test for the existence of a control.

Parameters

win - Window or dialog containing the control
id - Identifier of the control

Return value

1 if the control exists or 0 if it has not been created yet

Remarks

See Also: [CONTROL](#)

Example usage

```
IF CONTROLEXISTS(mydlg, 4) THEN SETCONTROLTEXT(mydlg,4,"hello")
```

12.78 COPYFILE

Syntax

INT = COPYFILE(source as STRING,dest as STRING,fail as INT)

Description

Copies the file specified by the source path to the destination path

Parameters

source - Source file to copy.

dest - Destination filename. The destination filename does not have to be the same as the source.
fail - If fail = 1 then the function will not copy the file if it already exists in the destination directory.
If fail = 0 then the file will be overwritten

Return value

COPYFILE returns 0 on error.

Remarks

See Also: [CREATEDIR](#)

Example usage

```
COPYFILE "c:\\mylog.log", "c:\\logs\\today.log"
```

12.79 COPYRGN

Syntax

UINT = COPYRGN(hrgn as UINT)

Description

Copies a Windows region

Parameters

hrgn - Handle to a Windows region.

Return value

A copy of the region

Remarks

A region should be deleted with [DELETERGN](#) if it is no longer in use. Unless the region is passed to a button of type @RGNBUTTON in which case the button takes care of freeing the region when it is destroyed.

Example usage

```
SETCONTROLCOLOR d1,BUTTON_1,RGB(255,255,255),RGB(10,100,128)  
SETBUTTONRGN d1,BUTTON_1,COPYRGN(hrgn)  
SEHTCOLOR d1,BUTTON_1,RGB(20,138,138)  
SETBUTTONBORDER d1,BUTTON_1,0
```

12.80 COS

Syntax

DOUBLE = COS(num as DOUBLE)

Description

Calculates the cosine of an angle.

Parameters

num - The angle in radians to calculate the cosine.

Return value

The cosine of the angle *num*.

Remarks

See Also [COSD](#), [FCOS](#), [FCOSD](#)

Example usage

```
PRINT COS(1.112)
```

12.81 COSD

Syntax

DOUBLE = COSD(num as DOUBLE)

Description

Calculates the cosine of an angle.

Parameters

num - The angle in degrees to calculate the cosine.

Return value

The cosine of the angle *num*.

Remarks

See Also [COS](#), [FCOS](#), [FCOSD](#)

Example usage

```
PRINT COSD(45.0)
```

12.82 COSH

Syntax

DOUBLE = COSH(num as DOUBLE)

Description

Calculates the hyperbolic cosine of an angle.

Parameters

num - The angle in radians to calculate the hyperbolic cosine

Return value

The hyperbolic cosine of the angle.

Remarks

See Also [COSHD](#), [FCOSH](#), [FCOSHD](#)

Example usage

```
PRINT COSH(1.112)
```

12.83 COSHD

Syntax

DOUBLE = COSHD(num as DOUBLE)

Description

Calculates the hyperbolic cosine of an angle.

Parameters

num - The angle in degrees to calculate the hyperbolic cosine

Return value

The hyperbolic cosine of the angle *num*.

Remarks

See Also [COSH](#), [FCOSH](#), [FCOSHD](#)

Example usage

```
PRINT COSHD(1.112)
```

12.84 CREATECOMOBJECT

Syntax

COMREF = CREATECOMOBJECT (name as STRING, machine as STRING)

Description

Creates a COM object using script syntax.

Parameters

name - name of the object to create.

machine - UNC of a remote machine.

Return value

Returns the COM object if successful, NULL otherwise.

Remarks

COM objects that run in their own process, such as Word and Excel can be created on a remote computer, assuming you have access rights.

Example usage

```
IDispatch Connection  
POINTER szResponse
```

```
INT _status
Connection = CreateComObject("MSXML2.XMLHTTP.3.0", "")
CallObjectMethod(Connection, ".Open(%s, %s, %b)", "GET", "http://myserver.com/test.xml")
CallObjectMethod(Connection, ".Send")
GetComProperty(Connection, "%d", &_status, ".status")
if _status = 200
    GetComProperty(Connection, "%T", &_szResponse, ".ResponseXML.xml")
    PRINT w2s(*<wstring>_szResponse)
    FreeComString(_szResponse)
endif
Connection->Release()
```

12.85 CREATEDIALOG

Syntax

CREATEDIALOG(dlg:DIALOG,x:INT,y:INT,w: INT,h:INT,style:UINT,parent:POINTER,
caption:STRING,proc:UINT)

Description

Defines a dialog to be used with the DOMODAL or SHOWDIALOG statements.

Parameters

dlg - A DIALOG variable.

x, y, w, h - The dialogs coordinates and dimensions.

style - Style flags for the dialog.

parent - A window or dialog to use as the parent, can be NULL.

caption - The title bar caption for the dialog

proc - Address of the subroutine that will handle the dialogs messages

Return value

None

Remarks

CREATEDIALOG creates a dialog template in memory. The dialog does not exist until DOMODAL or SHOWDIALOG is called. Initialize controls in response to the @IDINITDIALOG message.

See Also: [CLOSEDIALOG](#), [DOMODAL](#), [SHOWDIALOG](#), [CONTROL](#)

Example usage

```
DEF colordlg as DIALOG
CREATEDIALOG colordlg,0,0,317,255,0x80C80080,0,"Color Control Demo",&handler
```

12.86 CREATEDIR

Syntax

INT = CREATEDIR (path as STRING)

Description

Creates a directory

Parameters

path - full path to new directory

Return value

Returns 0 if the directory could not be created

Remarks

CREATEDIR can only create a directory below one that already exists. The path name should not contain a trailing slash.

Example usage

```
CREATEDIR "c:\\myfiles\\new directory"
```

12.87 CREATEMENU

Syntax

UINT = CREATEMENU(OPT popup as INT)

Description

Low level menu function used internally by menu creation macros.

Parameters

popup - Optional. Specifies whether the returned handle is a top level or popup menu. If equal to 1 then a popup menu is created

Return value

A handle to the newly created menu

Remarks

Used only for special cases where the menu creation macros do not provide enough functionality.

See Also: [APPENDMENU](#), [SETMENU](#)

Example usage

```
hMenu = CreateMenu()  
hPopup = CreateMenu(1)  
APPENDMENU(hMenu, "File", MF_POPUP|MF_STRING, hPopup)  
APPENDMENU(hPopup, "Open", MF_STRING, 1)  
APPENDMENU(hPopup, "Quit", MF_STRING, 2)  
SETMENU win, hMenu
```

12.88 CREATEREGKEY

Syntax

```
error =CREATEREGKEY(string key)
```

Description

Creates a registry key.

Parameters

key - The registry key path.

Return value

Returns 0 for no error, or 1 if the key could not be created

Remarks

None.

Example usage

```
Result=CreateRegKey("HKEY_CURRENT_USER\\Software\\XYZCorp\\File")
```

12.89 DATA

Syntax

```
DATA datum, ...
```

Description

Defines data within a data block.

Parameters

datum - One or more data items

Return value

None

Remarks

Currently IW BASIC supports INT, FLOAT, DOUBLE and STRING data items. Data will be stored contiguously in memory, there is no difference between multiple DATA statements and a single one except for readability.

See Also: [DATABEGIN](#), [DATAEND](#), [GETDATA](#), [RESTORE](#)

Example usage

```
DATABEGIN mydata  
DATA 1,2,3  
DATA 4,10,2002  
DATA "Monday","Tuesday","Wednesday",Thursday,"Friday"  
DATA "Saturday","Sunday"  
DATAEND
```

12.90 DATABEGIN

Syntax

DATABEGIN name

Description

Begins defining a block of data

Parameters

name - Name of the data block

Return value

None

Remarks

Each DATABEGIN statement must be paired with an DATAEND statement. The only command that can appear between the two are DATA statements.

See Also: [GETDATA](#), [DATAEND](#), [DATA](#), [RESTORE](#)

Example usage

```
DEF a as INT

RESTORE mydata

FOR x = 1 to 6
    GETDATA mydata,a
PRINT a
NEXT x

DO:UNTIL INKEY$ <> ""
END

DATABEGIN mydata
DATA 1,2,3
DATA 4,10,2002
DATAEND
```

12.91 DATAEND

Syntax

DATAEND

Description

Ends the definition of a data block

Parameters

None

Return value

None

RemarksSee Also: [GETDATA](#), [DATABEGIN](#), [DATA](#), [RESTORE](#)**Example usage**

```
DATABEGIN mydata
DATA 1,2,3
DATA 4,10,2002
DATAEND
```

12.92 DATE\$

Syntax

STRING = DATE\$(OPT strFormat as STRING)

Description

Returns the current date as a string

Parameters*strFormat* - Optional date formatting string. Default format is DD-MM-YYYY

Formatting specifiers:

Specifier	Result
d	Day of month as digits with no leading zero for single-digit days.
dd	Day of month as digits with leading zero for single-digit days.
ddd	Day of week as a three-letter abbreviation.
dddd	Day of week as its full name.
M	Month as digits with no leading zero for single-digit months.
MM	Month as digits with leading zero for single-digit months.
MMM	Month as a three-letter abbreviation.
MMMM	Month as its full name.
y	Year as last two digits, but with no leading zero for years less than 10.
yy	Year as last two digits, but with leading zero for years less than 10.
yyyy	Year represented by full four digits.

Return value

The current date

Remarks

To enclose text in the output string use single quote marks in the format string.

Example usage

```
PRINT DATE$
PRINT DATE$("ddd', ' MMM dd yy")
```

12.93 DateTimePicker

Syntax

UINT = DateTimePicker(win as WINDOW, title as STRING, l as INT, t as INT, w as INT, h as INT, flags as INT, exStyle as INT, id as UINT)

Description

Creates a DateTimePicker control.

Parameters

win - Parent window or dialog containing the control.

title - Title of the control.

l, t, w, h - Position and dimension of the control.

flags - Styles for the control.

exStyle - Extended window styles for the control.

id - Control identifier.

Return value

If used with a window the return value is the handle of the control. If used with a dialog the return value is 0 and the handle to the control can be obtained with GETCONTROLHANDLE during the @IDINITDIALOG message.

Remarks

The control can also be created using CONTROLEX with a window class of "SysDateTimePick32".

Styles

@DTS_UPDOWN - Use UPDOWN instead of MONTHCAL

@DTS_SHOWNONE - Allow a NONE selection

@DTS_SHORTDATEFORMAT - Use the short date format

@DTS_LONGDATEFORMAT - Use the long date format

@DTS_TIMEFORMAT - Use the time format

@DTS_APPCANPARSE - Allow user entered strings (app MUST respond to DTN_USERSTRING)

@DTS_RIGHTALIGN - right-align popup instead of left-align it

Example usage

```
DateTimePicker cp, "Pick a date", 40, 80, 140, 30, @BORDER|@TABSTOP|@DTS_TIMEFORMAT|@DTS_SHC
```

12.94 DEBUGPRINT

Syntax

DEBUGPRINT(strDebug as STRING)

Description

Outputs a string to the Debug view of the IDE.

Parameters

strDebug - String to output.

Return value

None.

Remarks

The function is ignored during a normal build. Even so you should still surround any DEBUGPRINT statements with \$IFDEF DEBUG / \$ENDIF pairs. The string is only displayed while in a debug session.

Example usage

```
$IFDEF DEBUG
    DEBUGPRINT "Program starting"
    DEBUGPRINT "A=" + STR$(A)
$ENDIF
error = MyFunction(27)
$IFDEF DEBUG
    DEBUGPRINT "MyFunction returned: " + STR$(error)
$ENDIF
```

12.95 DECLARE

Syntax

```
DECLARE {CDECL} localname(paramlist), returntype
DECLARE {CDECL} IMPORT, apiname {ALIAS realname}(paramlist), returntype
DECLARE {CDECL} EXTERN globalname(paramlist), returntype
```

*DECLARE labelname AS type

Description

Declares a local, external, or import function and its parameters.

*Declares an assembler label for tying an IWBasic variable to an assembly data segment entry.

Parameters

CDECL - Optional signifies the function uses the 'C' calling convention.

localname - Name of a local SUB in this source file.

apiname - Name of a function in a DLL.

globalname - Name of a global SUB in a different source file.

ALIAS realname - *apiname* is an alias for *realname*.

paramlist - Comma separated list of parameters in the format of *variable as type* or *variable: type*.

returntype - Type that function returns.

labelname - Name of a variable and a corresponding label in the assembly data segment.

type - *labelname* variable/data type

Return value

None

Remarks

IW BASIC uses a two pass compiler so it is not strictly necessary to declare a SUB only used in a single source file. It is supported both for backwards compatibility and readability.

The EXTERN keyword is used for multi module (multiple source file) programming. The subroutine must be marked as global using the GLOBAL statement in the source file it is contained in.

The parameter list is optional for local functions. For externals and import functions that have no parameters use an empty list ().

See Also: [SUB](#), [Assembly Data Segment](#)

Example usages

```
'An API declare
DECLARE IMPORT,ZeroMemory ALIAS RtlZeroMemory(pvoid as POINTER,length as INT),INT
'A local declare
DECLARE MyFunction(a as INT, b as STRING),STRING
'An external function
DECLARE EXTERN MyFunction2(ff as FLOAT), FLOAT
```

12.96 DEF / DIM

Syntax

DEF name as type

DEF name[...] as type

DIM name as type

DIM name[...] as type

Description

Defines a variable.

Parameters

name - Name of new variable.
type - Any built in type or UDT name.
... - Up to 3 array dimensions.

Return value

None

Remarks

Array dimensions must be a constant or resolve to a constant. For dynamic arrays see the NEW statement. A colon ':' may be used as a shortcut to the AS keyword. DIM is a synonym for DEF and the two can be used interchangeably.

You can also use the reverse definition method and eliminate the DEF / DIM statement.

See Also: [EXTERN](#), [NEW](#)

Example usage

```
DEF a[10] as INT
DEF b$ as STRING
DEF c[2,10] : WORD

'Reverse Definitions
INT a[10]
STRING b$
WORD c[2,10]
```

12.97 DEFAULT

Syntax

DEFAULT

Description

Defines a DEFAULT execution point if all the CASE statements in a SELECT statement are FALSE.

Parameters

None

Return value

None

Remarks

Any statements after the DEFAULT statement are executed when all CASE statements are FALSE. DEFAULT is optional.

See Also: [SELECT](#), [ENDSELECT](#), [CASE](#), [CASE&](#)

Example usage

```

SELECT A
  CASE 1
    PRINT "A = 1"
  CASE 2
    PRINT "A = 2"
  DEFAULT
    PRINT "None of the above"
ENDSELECT

```

12.98 DEFINE_GUID**Syntax**

DEFINE_GUID(lpGUID as GUID, l1 as UINT, w1 as WORD, w2 as WORD, b1 as CHAR, b2 as CHAR, b3 as CHAR, b4 as CHAR, b5 as CHAR, b6 as CHAR, b7 as CHAR, b8 as CHAR)

Description

Used to fill in a GUID UDT from individual members.

Parameters

l1 - A UINT value
w1, *w2* - Word values
b1 - *b8* - Byte values

Return value

None

Remarks

This function is used as a convenience as many Windows API headers use a similar macro to define a GUID.

See Also: [COM usage](#)

Example usage

```

DEF CLSID_DirectDraw as GUID
DEFINE_GUID( CLSID_DirectDraw, 0xD7B70EE0, 0x4340, 0x11CF, 0xB0, 0x63, 0x00, 0x20, 0xAF, 0xC2,

```

12.99 DELETE**Syntax**

DELETE(p as POINTER)

Description

Deletes a dynamic variable

Parameters

p - A pointer to a dynamic variable created with the NEW function

Return value

None

Remarks

See Also: [NEW](#)

Example usage

```
DEF p as POINTER
p = NEW(CHAR,1)
'...
DELETE p
```

12.100DELETEFILE

Syntax

INT = DELETEFILE(name as STRING)

Description

Deletes the file specified.

Parameters

name - Complete path to the file.

Return value

Returns 0 on failure.

Remarks

DELETEFILE can fail if the file is currently locked by the OS

Example usage

```
DELETEFILE "c:\\myfiles\\myfile.txt"
```

12.101DELETEIMAGE

Syntax

DELETEIMAGE(handle as UINT,iType as INT)

Description

Frees memory used by an image previously loaded with the LOADIMAGE statement. Type must be the same as specified in the LOADIMAGE statement.

Parameters

.handle - Handle to image returned by LOADIMAGE.

iType - Type of image handle points to.

Return value

None

Remarks

See Also: [LOADIMAGE](#)

Example usage

```
DELETEIMAGE myimage,@IMGBITMAP
```

12.102 DELETEREGKEY

Syntax

```
error =DELETEREGKEY(string key)
```

Description

Deletes a registry key.

Parameters

key - The registry key path.

Return value

Returns 0 for no error, or 1 if the key could not be deleted

Remarks

None.

Example usage

```
Result=DeleteRegKey("HKEY_CURRENT_USER\\Software\\XYZCorp\\File")
```

12.103 DELETERGN

Syntax

```
DELETERGN(hrgn as UINT)
```

Description

Deletes a region.

Parameters

hrgn - The region to be deleted.

Return value

None

Remarks

A region should be deleted if it is no longer in use. Unless the region is passed to a button of type @RGNBUTTON in which case the button takes care of freeing the region when it is destroyed.

Example usage

```
hrgn = RGNFROMBITMAP (GETSTARTPATH+"rgn_bmp1.bmp")

SETCONTROLCOLOR d1,BUTTON_1,RGB(255,255,255),RGB(10,100,128)
SETBUTTONRGN d1,BUTTON_1,COPYRGN(hrgn)
SEHTCOLOR d1,BUTTON_1,RGB(20,138,138)
SETBUTTONBORDER d1,BUTTON_1,0

SETFONT d1,"Arial",14,800,0,BUTTON_2
SETCONTROLCOLOR d1,BUTTON_2,RGB(255,255,255),RGB(10,100,128)
SETBUTTONRGN d1,BUTTON_2,COPYRGN(hrgn)
SETBUTTONBITMAPS d1,BUTTON_2,LOADIMAGE (GETSTARTPATH+"button_bmp_normal.bmp",@IMGBITMAI
SEHTCOLOR d1,BUTTON_2,RGB(80,80,80)

DELETERGN (hrgn)
```

12.104DELETESTRING

Syntax

DELETESTRING(win as WINDOW,id as UINT,pos as INT)

Description

Removes a string from a combobox or listbox control. Remaining strings are moved up to fill the empty position.

Parameters

win - Window or dialog containing the control.

id - Identifier of the control

pos - Zero based position of the string.

Return value

None

Remarks

See Also: [ADDSTRING](#)

Example usage

```
DELETESTRING mydlg, 7, 0
```

12.105DICTADD

Syntax

INT = DictAdd(dict as POINTER,key as STRING,value as STRING),

Description

Adds or replaces a key-value association..

Parameters

dict - Pointer to an associative array created with the DICTCREATE command.

key - The key string.

value - The string value.

Return value

TRUE on success, FALSE otherwise..

Remarks

If the associated element already exists then the value is replaced.

Example usage

```
mydict = DictCreate(100)

DictAdd(mydict, "John", "Red")
DictAdd(mydict, "Joe", "Green")
DictAdd(mydict, "Sue", "Yellow")
DictAdd(mydict, "Phil", "Purple")
DictAdd(mydict, "Jerry", "Orange")
DictAdd(mydict, "Lisa", "Blue")
DictAdd(mydict, "Tammy", "Black")
DictAdd(mydict, "George", "white")
DictAdd(mydict, "Paul", "Indigo")
DictAdd(mydict, "Merideth Abigail", "Violet")
```

12.106 DICTCREATE

Syntax

POINTER = DICTCREATE(OPT hashtablesize=17 as INT, OPT blocksize=10 as INT)

Description

Creates an associative array (dictionary) and initialize the hash table.

Parameters

hashtablesize - Number of entries in the hash table. For best performance, the hash table size should be a prime number. To minimize collisions the size should be roughly 20 percent larger than the largest anticipated data set. (optional default=17)

blocksize - Specifies the memory-allocation granularity for extending the hash entries. (optional default=10)

Return value

A pointer to an associative array..

Remarks

As the array grows, memory is allocated in units of blockSize entries.

Example usage

```
mydict = DictCreate(100)

DictAdd(mydict, "John", "Red")
```

```
DictAdd(mydict, "Joe", "Green")
DictAdd(mydict, "Sue", "Yellow")
DictAdd(mydict, "Phil", "Purple")
DictAdd(mydict, "Jerry", "Orange")
DictAdd(mydict, "Lisa", "Blue")
DictAdd(mydict, "Tammy", "Black")
DictAdd(mydict, "George", "white")
DictAdd(mydict, "Paul", "Indigo")
DictAdd(mydict, "Merideth Abigal", "Violet")
```

12.107 DICTFREE

Syntax

DICTFREE(dict as POINTER)

Description

Frees an associative array, removing all name/value pairs..

Parameters

dict - Pointer to an associative array created with the DICTCREATE command.

Return value

None.

Remarks

None

Example usage

```
DICTFREE(myDict)
```

12.108 DICTGETKEY

Syntax

STRING = DictGetKey(pAssoc as POINTER)

Description

Returns the key used to index the data in the array.

Parameters

pAssoc - A pointer returned by GetNextAssoc.

Return value

A string key.

Remarks

Use this method while iterating through the associative array.

Example usage

```

PRINT:PRINT "Iterating":PRINT "-----"
pos = DictGetStartAssoc(myDict)
WHILE pos
    pAssoc = DictGetNextAssoc(myDict,pos)
    PRINT DictGetKey(pAssoc)," ",DictGetValue(pAssoc)
ENDWHILE

```

12.109 DICTGETNEXTASSOC

Syntax

POINTER = DictGetNextAssoc(dict as POINTER,rNextPosition as UINT BYREF)

Description

Returns the absolute value of a number.

Parameters

dict - Pointer to an associative array created with the DICTCREATE command.

rNextPosition - Specifies a reference to a POSITION value returned by a previous GetNextAssoc or GetStartPosition command

Return value

A pointer to an association.

Remarks

Retrieves the array element at rNextPosition, then updates rNextPosition to refer to the next element in the array. This function is most useful for iterating through all the elements in the map. Note that the position sequence is not necessarily the same as the key value sequence.

Example usage

```

PRINT:PRINT "Iterating":PRINT "-----"
pos = DictGetStartAssoc(myDict)
WHILE pos
    pAssoc = DictGetNextAssoc(myDict,pos)
    PRINT DictGetKey(pAssoc)," ",DictGetValue(pAssoc)
ENDWHILE

```

12.110 DICTGETSTARTASSOC

Syntax

UINT = DictGetStartAssoc(dict as POINTER)

Description

Starts a map iteration by returning a POSITION value that can be passed to a GetNextAssoc command.

Parameters

dict - Pointer to an associative array created with the DICTCREATE command.

Return value

A POSITION value that indicates a starting position for iterating the array; or NULL if the array is empty.

Remarks

None

Example usage

```
PRINT:PRINT "Iterating":PRINT "-----"  
pos = DictGetStartAssoc(myDict)  
WHILE pos  
    pAssoc = DictGetNextAssoc(myDict,pos)  
    PRINT DictGetKey(pAssoc)," ",DictGetValue(pAssoc)  
ENDWHILE
```

12.111 DICTGETVALUE

Syntax

STRING = DictGetValue(pAssoc as POINTER)

Description

Returns the string value from an association.

Parameters

pAssoc - A pointer returned by GetNextAssoc.

Return value

The string value..

Remarks

None

Example usage

```
PRINT:PRINT "Iterating":PRINT "-----"  
pos = DictGetStartAssoc(myDict)  
WHILE pos  
    pAssoc = DictGetNextAssoc(myDict,pos)  
    PRINT DictGetKey(pAssoc)," ",DictGetValue(pAssoc)  
ENDWHILE
```

12.112 DICTLOOKUP

Syntax

STRING = DictLookup(dict as POINTER,key as STRING)

Description

Looks up the value associated with a given key.

Parameters

dict - Pointer to an associative array created with the DICTCREATE command.

key - The key string.

Return value

The string value associated with the given key.

Remarks

Lookup uses a hashing algorithm to quickly find the association element with a key that exactly matches the given key.

Example usage

```
PRINT DictLookup(myDict,"Lisa")
PRINT DictLookup(myDict,"Sue")
PRINT DictLookup(myDict,"Jerry")
```

12.113 DICTREMOVE

Syntax

INT = DictRemove(dict as POINTER, key as STRING)

Description

Looks up the array entry corresponding to the supplied key; then, if the key is found, removes the entry..

Parameters

dict - Pointer to an associative array created with the DICTCREATE command.

key - Key for the element to be removed

Return value

TRUE if entry was found and removed, FALSE otherwise.

Remarks

None

Example usage

```
DICTREMOVE myDict, "Jerry"
```

12.114 DICTREMOVEALL

Syntax

DictRemoveAll(dict as POINTER)

Description

Removes all entries from an associative array.

Parameters

dict - Pointer to an associative array created with the DICTCREATE command.

Return value

None.

Remarks

None

Example usage

```
DICTREMOVEALL myDict
```

12.115DO

Syntax

DO

Description

Begins a DO UNTIL loop.

Parameters

None

Return value

None

Remarks

To break out of a DO/UNTIL loop early you can use the [BREAK](#) command.

See Also: [UNTIL](#)

Example usage

```
DO  
UNTIL INKEY$ <> ""
```

12.116DOMODAL

Syntax

INT = DOMODAL(dlg as DIALOG, OPT parent as DIALOG|WINDOW)

Description

Shows a modal dialog.

Parameters

dlg - A DIALOG variable initialized with the CREATEDIALOG statement

parent - Optional. Specifies the parent window or dialog for this dialog. Overrides the parent

parameter in the CREATEDIALOG statement

Return value

The value given to the CLOSEDIALOG statement or @IDCANCEL

Remarks

See Also [CLOSEDIALOG](#), [CREATEDIALOG](#)

Example usage

```
DOMODAL d1, mainwin
```

12.117DOUBLE

Syntax

```
double = DOUBLE(num)
```

Description

Converts a numeric value to a double.

Parameters

num - A DOUBLE or FLOAT

Return value

If num is a DOUBLE then the return is an INT64 otherwise an INT.

Remarks

INT truncates the decimal portion. This behavior is different than assigning a floating point value to an integer variable where the rounding mode will return the largest integer.

Example usage

```
PRINT DOUBLE(1.56)
```

12.118DRAWMODE

Syntax

```
DRAWMODE(win as WINDOW,mode as INT)
```

Description

Sets the background fill mode. The background mode defines whether the system removes existing background colors on the drawing surface before drawing text or any non-solid line. Mode is either @TRANSPARENT or @OPAQUE

Parameters

win - The window to change drawing modes.

mode - The new drawing mode.

Return value

None

Remarks

@OPAQUE Background is filled with the current background color before the text or line is drawn. This is the default background mode.

@TRANSPARENT Background is not changed before drawing

Example usage

```
DRAWMODE mywnd, @TRANSPARENT
```

12.119dtpGetMCColor

Syntax

UINT = dtpGetMCColor(win as WINDOW, id as UINT, index as INT)

Description

Retrieves the color for a given portion of the month calendar within a date and time picker (DTP) control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Value specifying which color to retrieve.

Return value

The color in RGB format

Remarks

Valid values for index are:

- 0 The background color (between months)
- 1 The dates
- 2 Background of the title
- 3 Foreground of the title.
- 4 Background within the month calendar
- 5 The text color of header & trailing days

Example usage

```
Clr = dtpGetMCColor(demo, 200, 0)
```

12.120dtpGetSystemTime

Syntax

```
SYSTEMTIME = dtpGetSystemTime(win as WINDOW,id as UINT)
```

Description

Retrieves the currently selected time from a date and time picker (DTP) control and places it in a SYSTEMTIME UDT.

Parameters

None

Return value

A SYSTEMTIME UDT containing the currently selected time.

Remarks

None

Example usage

```
SYSTEMTIME st = dtpGetSystemTime(demo, 200)
word hour = st.wHour
word minute = st.wMinute
word second = st.wSecond
word day = st.wDay
word month = st.wMonth
word year = st.wYear
```

12.121dtpSetFormat

Syntax

```
dtpSetFormat(win as WINDOW,id as UINT,format as STRING)
```

Description

Sets the display of a date and time picker (DTP) control based on the given format string.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

format - The formatting string

Return value

None

Remarks

A DTP format string consists of a series of elements that represent a particular piece of information and define its display format. The elements will be displayed in the order they appear in the format string.

Date and time format elements will be replaced by the actual date and time.

Format string definitions:

Element	Description
"d"	The one- or two-digit day.
"dd"	The two-digit day. Single-digit day values are preceded by a zero.
"ddd"	The three-character weekday abbreviation.
"dddd"	The full weekday name.
"h"	The one- or two-digit hour in 12-hour format.
"hh"	The two-digit hour in 12-hour format. Single-digit values are preceded by a zero.
"H"	The one- or two-digit hour in 24-hour format.
"HH"	The two-digit hour in 24-hour format. Single-digit values are preceded by a zero.
"m"	The one- or two-digit minute.
"mm"	The two-digit minute. Single-digit values are preceded by a zero.
"M"	The one- or two-digit month number.
"MM"	The two-digit month number. Single-digit values are preceded by a zero.
"MMM"	The three-character month abbreviation.
"MMMM"	The full month name.
"t"	The one-letter AM/PM abbreviation (that is, AM is displayed as "A").
"tt"	The two-letter AM/PM abbreviation (that is, AM is displayed as "AM").
"yy"	The last two digits of the year (that is, 1996 would be displayed as "96").
"yyyy"	The full year (that is, 1996 would be displayed as "1996").

Example usage

```
dtpSetFormat demo,200,"dd-MMM-yyyy HH:mm"
```

12.122dtpSetMCColor

Syntax

```
dtpSetMCColor(win as WINDOW,id as UINT,index as INT,clr as UINT)
```

Description

Sets the color for a given portion of the month calendar within a date and time picker (DTP) control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.
index - Value specifying which color to set.
clr - The color in RGB format.

Return value

None

Remarks

Valid values for index are:

- 0 The background color (between months)
- 1 The dates
- 2 Background of the title
- 3 Foreground of the title.
- 4 Background within the month calendar
- 5 The text color of header & trailing days

Example usage

```
ntpSetMCColor demo, 200, 4, RGB(200,200,200)
```

12.123ntpSetSystemTime

Syntax

```
ntpSetSystemTime(win as WINDOW,id as UINT,tm as SYSTEMTIME)
```

Description

Sets a date and time picker (DTP) control to a given date and time.

Parameters

win - Dialog or window containing the control.
id - Identifier of the control.
tm - A SYSTEMTIME UDT containing the date and time.

Return value

None

Remarks

None

Example usage

```
SYSTEMTIME st
st.wHour = 1
st.wMinute = 30
st.wSecond = 0
st.wDay = 25
st.wMonth = 12
st.wYear = 2008
ntpSetSystemTime demo,200,st
```

12.124EACH

Syntax

FOR pData = EACH pList {AS type}

Description

Reserved word. Used in a FOR/EACH loop

Parameters

None

Return value

None

Remarks

See Also: [FOR](#)

Example usage

```
FOR mydata = EACH mylist AS STRING
  PRINT #mydata
NEXT
```

12.125ELLIPSE

Syntax

ELLIPSE(win as WINDOW, left:INT, top:INT, width:INT, height:INT, OPT outline as UINT, OPT fill as UINT)

Description

Draws an ellipse in window bound by the rectangle specified by left, top, width and height.

Parameters

win - Window to draw the ellipse in.

left, top, width, height - Bounding rectangle for the ellipse.

outline - Optional outline color.

fill - Optional fill color.

Return value

None

Remarks

If *outline* and *fill* are not specified the ellipse will be drawn in the current foreground color.

Example usage

```
ELLIPSE mywnd, 10,10,100,100, RGB(255,0,0), RGB(10,10,128)
```

12.126ELSE

Syntax

ELSE

Description

Reserved word. Used in IF statements and IF blocks.

Parameters

None

Return value

None

Remarks

See Also: [IF](#)

Example usage

```
IF a$ = "hello" THEN PRINT "goodbye" ELSE PRINT "hello"
```

12.127ELSEIF

Syntax

ELSEIF condition

Description

Reserved word. Used in IF statements and IF blocks.

Parameters

Condition - Condition to test for continuing execution.

Return value

None

Remarks

See Also: [IF](#)

Example usage

```
IF name = "Jerry"  
    pay = "7.55"  
ELSEIF name = "Tom"  
    pay = "9.55"  
ELSEIF name = "Lisa"  
    pay = "10.34"  
ENDIF
```

12.128 ENABLECONTROL

Syntax

ENABLECONTROL(win as WINDOW,id as UINT,bEnable as INT)

Description

Enables or disables a control in a window or dialog.

Parameters

win - Window or dialog containing the control.

id - Identifier of the control.

bEnable - New state of the control. Set to 0 to disable and 1 to enable the control.

Return value

None

Remarks

None

Example usage

```
ENABLECONTROL mydlg, 7, 1
```

12.129 ENABLEMENU

Syntax

ENABLEMENU(win as WINDOW,pos as UINT,bEnable as INT)

Description

Enables or disables a top level menu (title). Position is the 0 based index of the menu.

Parameters

win - Window or dialog containing the menu.

pos - Position of the menu.

bEnable - New enable state of the menu. Use 0 to disable and 1 to enable.

Return value

None

Remarks

Disabling a menu title makes it not selectable.

Example usage

```
ENABLEMENU mywnd, 0, 0
```

12.130 ENABLEMENUITEM

Syntax

ENABLEMENUITEM(win as WINDOW, id as UINT, bEnable as INT)

Description

Enables or disables a menu item.

Parameters

win - Window or Dialog containing the menu

id - Menu item identifier

bEnable - Use 0 to disable or 1 to enable.

Return value

None

Remarks

Disabled menus are grayed-out and cannot be selected

See Also: [ENABLEMENU](#)

Example usage

```
ENABLEMENUITEM mywnd, 72, 1
```

12.131 ENABLETABS

Syntax

ENABLETABS(win as WINDOW, bEnable as INT)

Description

Allows windows to have the same shortcut processing as a dialog. When enabled the tab key will cycle through controls with the @TABSTOP style.

Parameters

win - Window to enable shortcut processing.

bEnable - Use 1 to enable and 0 to disable.

Return value

None

Remarks

Has no effect on dialogs.

Example usage

```
ENABLETABS mywnd, 1
```

12.132END

Syntax

END

Description

Ends your program.

Parameters

None

Return value

None

Remarks

END will cause the on-exit list to be iterated. Set a return value by use SETEXITCODE.

See Also: [ONEXIT](#), [SETEXITCODE](#)

Example usage

None

12.133ENDCATCH

Syntax

ENDCATCH

Description

The end of a block of code that handles exceptions from a TRY block..

Parameters

None

Return value

None

Remarks

See Also: [CATCH](#), [Exception Handling](#)

Example usage

```
exec_a:
  try
    print 1/a
  endtry
  catch
    a=1
    goto exec_a
  endcatch
```

12.134 ENDENUM

Syntax

ENDENUM

Description

Ends the definition of enumerated constants.

Parameters

None.

Return value

None

Remarks

Each enumeration value is sequentially numbered, starting from 1. You can override the starting value of any enumerated constant. The enumeration name becomes a type define for INT, so it can be used as a parameter to a subroutine, or defined like any other variable type.

The enumeration of constants is terminated with the ENDENUM statement.

Example usages

```
ENUM days
  monday
  tuesday
  wednesday
  thursday
  friday
  saturday
  sunday
ENDENUM

def payday as days
payday = thursday

ENUM dialog_messages
  WM_FUDGE = WM_USER
  WM_ICECREAM
  WM_CHOCOLATE
ENDENUM

ENUM colors
  red = 27, green, blue, orange, purple
ENDENUM
```

12.135ENDIF

Syntax

ENDIF

Description

Ends an IF block.

Parameters

None

Return value

None

Remarks

Every IF block must be terminated with an associated ENDIF statement

See Also: [IF](#)

Example usage

```
IF a = 4
    b = b + 1
ELSE
    b = b - 1
ENDIF
```

12.136ENDINTERFACE

Syntax

ENDINTERFACE

Description

Ends definition of a COM interface

Parameters

None

Return value

None

Remarks

Each INTERFACE statement must be paired with an ENDINTERFACE.

See Also: [INTERFACE](#), [SET_INTERFACE](#), [STDMETHOD](#), [DEFINE_GUID](#)

Example usage

```

INTERFACE IUnknown
    STDMETHOD QueryInterface(riid as POINTER, ppvObj as POINTER),INT
    STDMETHOD AddRef(),INT
    STDMETHOD Release(),INT
ENDINTERFACE

```

12.137 ENDMENU

Syntax

ENDMENU

Description

Ends a menu definition macro and sets the menu in the window or dialog.

Parameters

None

Return value

None

Remarks

Every BEGINMENU, BEGININSERTMENU, or CONTEXTMENU must be paired with an ENDMENU statement.

See Also: [BEGINMENU](#), [MENUITEM](#), [MENUTITLE](#), [BEGINPOPUP](#), [ENDPOPUP](#), [SEPARATOR](#), [BEGININSERTMENU](#), [CONTEXTMENU](#)

Example usage

BEGINMENU win
MENUTITLE "&File"
MENUITEM "Open",0,1
MENUITEM "Close",0,2
BEGINPOPUP "Save As..."
MENUITEM "Ascii",0,3
MENUITEM "Binary",0,4
ENDPOPUP
SEPARATOR
MENUITEM "&QUIT",0,5
MENUTITLE "&Edit"
MENUITEM "Cut",0,6
ENDMENU

12.138 ENDPAGE

Syntax

ENDPAGE(handle as UINT)

Description

Ends the current page and starts the next. On most printers this will eject the page.

Parameters

handle - Handle returned by the OPENPRINTER statement.

Return value

None

Remarks

See Also: [OPENPRINTER](#), [WRITERPRINTER](#), [CLOSEPRINTER](#)

Example usage

```
ENDPAGE hPrinter
```

12.139ENDPOPUP

Syntax

```
ENDPOPUP
```

Description

.Ends definition of a popup menu.

Parameters

None

Return value

None

Remarks

See Also: [BEGINPOPUP](#), [BEGINMENU](#), [ENDMENU](#), [MENUITEM](#), [MENUTITLE](#), [SEPARATOR](#), [BEGININSERTMENU](#)

Example usage

```
BEGINMENU win
  MENUTITLE "&File"
  MENUITEM "Open",0,1
  MENUITEM "Close",0,2
  BEGINPOPUP "Save As..."
    MENUITEM "Ascii",0,3
    MENUITEM "Binary",0,4
  ENDPUPUP
  SEPARATOR
  MENUITEM "&QUIT",0,5
  MENUTITLE "&Edit"
  MENUITEM "Cut",0,6
ENDMENU
```

12.140ENDSELECT

Syntax

ENDSELECT

Description

Marks the end of a SELECT block.

Parameters

None

Return value

None

Remarks

Every SELECT statement must be paired with an ENDSELECT.

See Also: [SELECT](#), [CASE](#), [CASE&](#), [DEFAULT](#)

Example usage

```
SELECT A
  CASE 1
    PRINT "TRUE!"
  CASE 2
    PRINT "You wont see this text"
  DEFAULT
    PRINT "None of the above"
ENDSELECT
```

12.141ENDSUB

Syntax

ENDSUB

Description

Marks the end of a subroutine.

Parameters

None

Return value

None

Remarks

Every SUB statement must be paired with an ENDSUB to mark the start and end of the subbroutine.

See Also: [SUB](#), [RETURN](#), [DECLARE](#)

Example usage

```
SUB IsDigit(c as CHAR)
```

```
SELECT 1
  CASE c < 0x30
    RETURN 0
  CASE c > 0x39
    RETURN 0
  default:
    RETURN 1
ENDSELECT
ENDSUB
```

12.142ENDTRY

Syntax

```
ENDTRY
```

Description

The end of a block of code that traps exceptions..

Parameters

None

Return value

None

Remarks

See Also: [TRY](#), [Exception Handling](#)

Example usage

```
exec_a:
  try
    print 1/a
  endtry
  catch
    a=1
    goto exec_a
  endcatch
```

12.143ENDTYPE

Syntax

```
ENDTYPE
```

Description

Ends the definition of a user defined data type (UDT).

Parameters

None

Return value

None

Remarks

See Also: [TYPE](#)

Example usage

```
TYPE foo
  DEF name as STRING
  DEF age as INT
ENDTYPE

DEF record as foo

record.name = "John Smith"
```

12.144ENDUNION

Syntax

ENDUNION

Description

Ends the definition of a union.

Parameters

None

Return value

None

Remarks

See Also: [UNION](#)

Example usage

```
TYPE MYDATA
  UNION u ' optional name
    TYPE personal
      string name      ' mydata.u.personal.name
      string surname
    ENDTYPE
    TYPE business
      string name
      int account
    ENDTYPE
  ENDUNION
ENDTYPE

DEF x as MYDATA
x.u.personal.name = "John"
x.u.personal.surname = "Doe"

x.u.business.name = "Acme Bolt"
x.u.business.account = 123456
```

12.145ENDWHILE

Syntax

ENDWHILE

Description

Tests the condition of the corresponding WHILE statement and if true will loop back to the next line of code following the WHILE statement

Parameters

None

Return value

None

Remarks

WEND is a synonym for ENDWHILE and can be used interchangeably.

See Also: [WHILE](#), [WEND](#)

Example usage

```
WHILE a < 7
  a = a + 1
ENDWHILE
```

12.146ENUM

Syntax

ENUM name

Description

Begins definition of enumerated constants.

Parameters

name - Name of the enumeration

Return value

None

Remarks

Each enumeration value is sequentially numbered, starting from 1. You can override the starting value of any enumerated constant. The enumeration name becomes a type define for INT, so it can be used as a parameter to a subroutine, or defined like any other variable type.

Basic math operations are allowed to calculate a constants value. The following operators may be

used: =, ==, <>, !=, +, -, *, /, %, |, OR, || (xor), &, &&, AND, <<, >>, !, ~, ^, >, >=, <, <=.

Since the final result must resolve to a fixed numeric value you cannot use functions or variables when calculating constants. You can however use other constant identifiers in the calculations.

The enumeration of constants is terminated with the ENDENUM statement.

Example usages

```
ENUM days
monday
tuesday
wednesday
thursday
friday
saturday
sunday
ENDENUM

def payday as days
payday = thursday

ENUM dialog_messages
  WM_FUDGE = WM_USER
  WM_ICECREAM
  WM_CHOCOLATE
ENDENUM

ENUM colors
  red = 27, green, blue, orange, purple
ENDENUM

enum myenum
  minusone = -1
  zero
  one
  two = one+1
  six = 2+2*2
endenum
```

12.147EOF

Syntax

INT = EOF(file)

Description

Tests an open file for the end of file condition.

Parameters

file - A FILE or BFILE variable successfully initialized with the OPENFILE command.

Return value

1 if the EOF has been reached or 0 if there is still data available to be read.

Remarks

See Also: [OPENFILE](#), [CLOSEFILE](#), [READ](#), [WRITE](#)

Example usage

```
DO
  READ file1,ln
UNTIL EOF(file1)
```

12.148EXP

Syntax

DOUBLE = EXP(num as DOUBLE)

Description

Calculates the exponential value of a number

Parameters

num - The number to calculate the exponential value

Return value

The exponential value of the number *num*

Remarks

None

Example usage

```
PRINT EXP(10)
```

12.149EXPORT

Syntax

EXPORT function_name

Description

Exports a function in a DLL build.

Parameters

function_name - The name of the subroutine .

Return value

None

Remarks

When creating a DLL use EXPORT to indicate which functions are available in the public export table. These functions will be visible to be imported in other programs and languages. An exported subroutine is automatically declared as GLOBAL. Subroutines not exported will be private to your

DLL.

See Also: [GLOBAL](#)

Example usage

```
export MYFUNCTION
export MYFUNCTION2
export INTRAND

SUB MYFUNCTION(in as INT),INT
RETURN in
ENDSUB

SUB MYFUNCTION2(),INT
RETURN 0
ENDSUB

SUB INTRAND(min as INT,max as INT),INT
RETURN RAND(min,max)
ENDSUB
```

12.150EXTERN

Syntax

EXTERN definition

Description

Defines a variable as being external, or existing in a different source file. Used in project builds.

Parameters

definition - Same variable definition format as the DEF / DIM statement

Return value

None

Remarks

The EXTERN definition must match the original definition. Variable names are case sensitive across source modules. EXTERN is also a keyword used in DECLARE statements.

See Also: [GLOBAL](#), [DECLARE](#), [DEF / DIM](#)

Example usage

```
EXTERN myvariable as UINT
```

12.151FACOS

Syntax

FLOAT = FACOS(num as FLOAT)

Description

Calculates the arccosine of an number.

Parameters

num - The number to take the arccosine

Return value

The angle in radians.

Remarks

Uses FLOAT precision instead of DOUBLE for time critical applications.

See also [ACOSD](#), [ACOS](#), [FACOSD](#)

Example usage

```
myacos = FACOS (FCOS (1.223))
```

12.152FACOSD

Syntax

FLOAT = FACOSD(num as FLOAT)

Description

Calculates the Arccosine of an number.

Parameters

num - The number to calculate the arccosine of

Return value

The angle in degrees.

Remarks

Uses FLOAT precision instead of DOUBLE for time critical applications.

See also [ACOSD](#), [ACOS](#), [FACOS](#)

Example usage

```
myacos = FACOSD (.8665)
```

12.153FASIN

Syntax

FLOAT = FASIN(num as FLOAT)

Description

Calculates the arcsine of a number.

Parameters

num - The number to calculate the arcsine.

Return value

The angle in radians of the arcsine of *num*.

Remarks

Uses FLOAT parameter and return for time critical applications.

See Also: [ASIND](#), [ASIN](#), [FASIND](#)

Example usage

```
PRINT FASIN(.86565)
```

12.154 FASIND

Syntax

FLOAT = FASIND(num as FLOAT)

Description

Calculates the arcsine of a number.

Parameters

num - The number to calculate the arcsine.

Return value

The angle in degrees of the arcsine of *num*.

Remarks

Uses FLOAT precision instead of DOUBLE for time critical applications.

See Also: [ASIND](#), [FASIN](#), [ASIN](#)

Example usage

```
PRINT FASIND(FSIND(45.0))
```

12.155 FATAN

Syntax

FLOAT = FATAN(num as FLOAT)

Description

Calculates the arctangent of a number.

Parameters

num - The number to calculate the arctangent.

Return value

The angle in radians of the arctangent of *num*.

Remarks

Uses FLOAT precision instead of DOUBLE for time critical applications.

See Also: [ATAND](#), [ATAN](#), [FATAN](#)

Example usage

```
PRINT FATAN(1.113)
```

12.156FATAND

Syntax

FLOAT = FATAND(num as FLOAT)

Description

Calculates the arctangent of a number.

Parameters

num - The number to calculate the arctangent..

Return value

The angle in degrees of the arctangent of *num*.

Remarks

Uses FLOAT precision instead of DOUBLE for time critical applications.

See Also: [ATAN](#), [ATAND](#), [FATAN](#)

Example usage

```
PRINT FATAND(FTAND(47.0))
```

12.157FCOS

Syntax

FLOAT = FCOS(num as FLOAT)

Description

Calculates the cosine of an angle.

Parameters

num - The angle in radians to calculate the cosine.

Return value

The cosine of the angle *num*.

Remarks

Uses FLOAT parameter and return for time critical applications

See Also [COS](#), [COSD](#), [FCOSD](#)

Example usage

```
PRINT FCOS(1.112)
```

12.158FCOSD

Syntax

FLOAT = FCOSD(num as FLOAT)

Description

Calculates the cosine of an angle.

Parameters

num - The angle in degrees to calculate the cosine

Return value

The cosine of the angle *num*.

Remarks

Uses FLOAT precisions for parameter and return for time critical applications

See Also [COS](#), [FCOS](#), [COSD](#)

Example usage

```
PRINT FCOSD(45.0)
```

12.159FCOSH

Syntax

FLOAT = FCOSH(num as FLOAT)

Description

Calculates the hyperbolic cosine of an angle.

Parameters

num - The angle in radians to calculate the hyperbolic cosine

Return value

The hyperbolic cosine of the angle *num*.

Remarks

Uses FLOAT parameter and return for time critical applications.

See Also [COSHD](#), [COSH](#), [FCOSHD](#)

Example usage

```
PRINT FCOSH(1.112)
```

12.160FCOSHD

Syntax

FLOAT = FCOSHD(num as FLOAT)

Description

Calculates the hyperbolic cosine of an angle.

Parameters

num - The angle in degrees to calculate the hyperbolic cosine.

Return value

The hyperbolic cosine of the angle *num*.

Remarks

Uses FLOAT parameter and return for time critical applications.

See Also [COSH](#), [FCOSH](#), [COSHD](#)

Example usage

```
PRINT FCOSHD(1.112)
```

12.161FILE

Syntax

file = FILE(num)

Description

Converts a value to a file.

Parameters

num - A UINT

Return value

If num is not a file pointer the results will be unpredictable.

Remarks**Example usage**

```
FILE (fptr)
```

12.162FILEREQUEST

Syntax

STRING = FILEREQUEST(title as STRING,parent as POINTER,nOpen as INT,OPT filter as STRING,OPT ext as STRING,OPT flags as INT,OPT initdir as STRING)

Description

Opens the standard file dialog and returns a string containing the fully qualified pathname to the user-selected file. For a 'Save as' dialog *nOpen* should be 0. For an 'Open' type dialog *nOpen* should be 1.

Parameters

title - Title of the file dialog
parent - Parent window or dialog, can be NULL
nOpen - Open dialog specifier.
filter - Filter string
ext - Default extension
flags - Additional style flags
initdir - Initial directory

Return value

A string containing the full path name to the file or a list of files.

Remarks

The optional filter variable limits the dialog to showing only certain file types. The string consists of ordered pairs separated by the '|' character and ending with two ||.

Example:

```
Filter$ = "Text files|.txt|All Files|.*)"
```

The default extension string should not contain a '.' and will be used in the 'Save as' dialog if the user does not type an extension.

The only additional style is @MULTISELECT in which case the return string will contain a list of files separated by the bar character '|'. If only one file is selected it will still contain a terminating bar '|'.

See Also: [COLORREQUEST](#), [FONTREQUEST](#)

Example usage

```
REM Define a buffer to hold the returned filenames.
DEF filenames[10000]:ISTRING
DEF filter,filetemp:STRING
DEF pos:int
filter = "All Files (*.*)|.*)"|Text Files (*.txt)|*.txt||"

filenames = filerequest("Select Multiple Files",0,1,filter,"txt",@MULTISELECT, "C:\\")
do
  pos = instr(filenames,"|")
  if(pos)
    filetemp = left$(filenames,pos-1)
    filenames = mid$(filenames,pos+1)
```

```
        REM do something with the file in filetemp
    endif
until pos = 0
```

12.163 FINDCLOSE

Syntax

FINDCLOSE(handle as UINT)

Description

After you are finished reading a directory you must close the handle with FINDCLOSE. If the handle is not closed memory loss will occur.

Parameters

handle - Handle returned by the FINDOPEN function

Return value

None

Remarks

You must not close a handle more than once.

See Also: [FINDOPEN](#), [FINDNEXT](#)

Example usage

```
OPENCONSOLE
DEF dir:UINT
DEF filename:STRING

dir = FINDOPEN("c:\\*.*.*)
IF(dir)
    DO
        filename = FINDNEXT(dir)
        PRINT filename
    UNTIL filename = ""
    FINDCLOSE dir
ENDIF

PRINT "Press Any Key"
DO:UNTIL INKEY$ <> ""

CLOSECONSOLE
END
```

12.164 FINDNEXT

Syntax

STRING = FINDNEXT(handle as UINT, OPT attrib as POINTER)

Description

After a directory is successfully opened with the FINDOPEN function use FINDNEXT to retrieve the filenames in a loop.

Parameters

handle - [in] Value returned by the FINDOPEN function.

attrib - [out] Optional. The files attributes stored as an integer value.

Return value

A string containing the filename, or an empty string "" when all files have been read.

Remarks

Returned filename does not include the path. The optional attrib parameter must be defined as an integer type and will receive the returned files attributes. The attributes are a bit mask that can contain one or more of the following:

```
@FILE_ARCHIVE
@FILE_COMPRESSED
@FILE_DEVICE
@FILE_DIRECTORY
@FILE_ENCRYPTED
@FILE_HIDDEN
@FILE_NORMAL
@FILE_READONLY
@FILE_SYSTEM
```

See Also: [FINDOPEN](#), [FINDCLOSE](#)

Example usage

```
OPENCONSOLE
DEF dir:UINT
DEF filename:STRING

dir = FINDOPEN("c:\\*.*.*)
IF(dir)
  DO
    filename = FINDNEXT(dir)
    PRINT filename
  UNTIL filename = ""
  FINDCLOSE dir
ENDIF

PRINT "Press Any Key"
DO:UNTIL INKEY$ <> ""

CLOSECONSOLE
END
```

Example returning only directories:

```
OPENCONSOLE
DEF dir,attrib:UINT
DEF filename:STRING
```

```
dir = FINDOPEN("c:\\*.*)
IF(dir)
  DO
    filename = FINDNEXT(dir,attrib)
    IF attrib & @FILE_DIRECTORY THEN PRINT filename
    UNTIL filename = ""
    FINDCLOSE dir
  ENDIF

PRINT "Press Any Key"
DO:UNTIL INKEY$ <> ""

CLOSECONSOLE
END
```

12.165 FINDOPEN

Syntax

UINT = FINDOPEN(dir as STRING)

Description

Opens a directory for reading filenames.

Parameters

dir - The directory to iterate filenames. Use wildcards to limit returned names.

Return value

Handle to open directory or 0 if the directory could not be opened for reading

Remarks

dir is a string that contains the full path to the directory plus any wildcard symbols for file matching. Example: "c:\\windows*.txt". Use *.* to retrieve a list of all the files.

See Also: [FINDNEXT](#), [FINDCLOSE](#)

Example usage

```
OPENCONSOLE
DEF dir:UINT
DEF filename:STRING

dir = FINDOPEN("c:\\*.*)
IF(dir)
  DO
    filename = FINDNEXT(dir)
    PRINT filename
    UNTIL filename = ""
    FINDCLOSE dir
  ENDIF

PRINT "Press Any Key"
DO:UNTIL INKEY$ <> ""

CLOSECONSOLE
```

END

12.166 FLOODFILL

Syntax

FLOODFILL(win as WINDOW, x as INT, y as INT, crColor as UINT)

Description

Fills an area containing the point x,y with the specified color. Filling continues outward until a color other than the one specified is encountered.

Parameters

win - Window.

x, y - Point to start the flood fill.

crColor - Color to fill.

Return value

None

Remarks

See Also: [RGB](#)

Example usage

```
FLOODFILL mywnd, 10, 20, RGB(255,0,0)
```

12.167 FLOOR

Syntax

DOUBLE = FLOOR(num as DOUBLE)

Description

The FLOOR function returns the largest integer that is less than or equal to the input parameter

Parameters

num - Number to test

Return value

A DOUBLE value containing the largest integer

Remarks

See Also: [CEIL](#)

Example usage

```
PRINT FLOOR(2.8), FLOOR(-2.8)
```

12.168FLT / FLOAT

Syntax

DOUBLE = FLT(num as INT64)

DOUBLE = FLOAT(num as INT64)

Description

Converts an integer to a floating point number for convenience in calculations.

Parameters

num - Integer number to convert.

Return value

num converted to a floating point type.

Remarks

Supports integers up to 64 bits.

See Also: [INT\(\)](#)

Example usage

```
INT a,b
a = 1:b=2
PRINT a/FLT(b)

DO:UNTIL INKEY$ <> ""
```

12.169FONTREQUEST

Syntax

STRING = FONTREQUEST(win as WINDOW,vSize as INT BYREF,vWeight as INT BYREF,vFlags as UINT BYREF,vColor as UINT BYREF,OPT name as STRING)

Description

The FONTREQUEST function opens the standard system font dialog. The function returns the name of the font and sets four variables with the attributes of the requested font.

Parameters

win - [in] Parent window or dialog

vSize - [in][out] The size of the selected font.

vWeight - [in][out] The weight of the font.

vFlags = [in][out] Attributes of the selected font

vColor = [in][out] Color of the selected font

name - [in] Optional font name to preset the combobox list.

Return value

.String containing the name of the selected font or an empty string, "", if the dialog was canceled.

Remarks

The four variables *vSize*, *vWeight*, *vFlags* and *vColor* must be defined as type INT. They should be set to the initial selections to be displayed to the user or 0 for default values.

See Also: [SETFONT](#)

Example usage

```
DEF size,weight,flags,col:INT
DEF fontname:STRING
...
fontname = FONTREQUEST(win,size,weight,flags,col)
IF fontname <> ""
    SETFONT win,fontname,size,weight,flags
    FRONTPEN win,col
ENDIF
```

12.170FOR

Syntax

FOR variable = start TO end OPT step
FOR pData = EACH pList {AS type}

Description

The first form of FOR loops from *start* to *end*. The second iterates through a linked list.

Parameters

variable - Integer counter variable. Must be of type CHAR, INT, UINT, or WORD

start - Starting count of the loop.

end - Ending count of the loop.

step - Optional stepping value. Defaults to 1.

pData - Pointer to receive each data pointer of the list in turn.

pList - Linked list to iterate.

type - Automatic typecasting name.

Return value

N/A

Remarks

step can be a negative value in which case the loop will count backwards. The number of loops is inclusive of the *end* value so a start of 0 and an end of 10 will loop 11 times. To break out of a FOR/NEXT loop early you can use the BREAKFOR or BREAK command.

The automatic typecasting name alleviates the need to use typecasting or the SETTYPE command on the returned data pointer.

See Also: [NEXT](#), [BREAKFOR](#), [BREAK](#)

Example usages

```
FOR x = 1 to 10
PRINT x
NEXT x

FOR mydata = EACH mylist AS STRING
PRINT #mydata
NEXT
```

12.171FreeHeap

Syntax

INT = FreeHeap(lpMem as UINT)

Description

Used internally by string functions. Frees memory allocated with AllocHeap

Parameters

lpMem - Memory handle returned by the AllocHeap function.

Return value

0 on failure

Remarks

This function should only be used by command implementers.

See Also: [AllocHeap](#)

Example usage

```
DEF p as POINTER
p = AllocHeap(100)
#<STRING>p = "Copied to heap memory"
PRINT #<STRING>p
FreeHeap(p)
```

12.172FREELIB

Syntax

FREELIB(name as STRING)

Description

Unloads a dynamically loaded DLL.

Parameters

name - The name or complete path the the DLL. If the DLL loaded was located in the system file path then it is not necessary to supply a complete pathname.

Return value

None

Remarks

Included for backwards compatibility with other languages

The function will only unload DLLs that were loaded by the calling process. An example of this would be the LoadLibrary API function. You can also unload an import library if no other part of your code attempts to use any function in the DLL.

Example usage

```
FREELIB "setup.dll"
```

12.173 FREEMEM

Syntax

FREEMEM(mem as MEMORY)

Description

Frees memory previously allocated with the ALLOCMEM statement.

Parameters

mem - A variable of type MEMORY used as a placeholder to access the allocated memory

Return value

None

Remarks

Once the memory is freed it is returned to the system and cannot be accessed again.

See Also: [ALLOCMEM](#)

Example usage

```
DEF mymem as MEMORY
IF ALLOCMEM(mymem,100,10)
    PRINT "Allocated 1000 bytes of memory"
    FREEMEM mymem
ENDIF
```

12.174 FRONTPEN

Syntax

FRONTPEN(win as WINDOW,crColor as UINT)

Description

Sets the foreground color for drawing operations in the window.

Parameters

win - Window to change the foreground drawing color
crColor - New foreground drawing color

Return value

None

Remarks

The color chosen by the FRONTPEN will be used by text, lines, outlines of rectangles, outlines of ellipses, and borders.

See Also: [BACKPEN](#)

Example usage

```
FRONTPEN mywnd, RGB(0,0,128)
```

12.175FSIN

Syntax

FLOAT = FSIN(num as FLOAT)

Description

Calculates the sine of an angle.

Parameters

num - The angle in radians to calculate the sine.

Return value

The sine of the angle *num*.

Remarks

Uses FLOAT parameter and return for time critical applications

See Also: [SIN](#), [SIND](#), [FSIND](#)

Example usage

```
PRINT FSIN(1.21)
```

12.176FSIND

Syntax

FLOAT = FSIND(num as FLOAT)

Description

Calculates the sine of an angle.

Parameters

num - The angle in degrees to calculate the sine.

Return value

The sine of the angle *num*.

Remarks

Uses FLOAT parameter and return for time critical applications

See Also: [SIN](#), [SIND](#), [FSIN](#)

Example usage

```
PRINT FSIND(45.0)
```

12.177FSINH

Syntax

FLOAT = FSINH(num as FLOAT)

Description

Calculates the hyperbolic sine of an angle.

Parameters

num - The angle in radians to calculate the hyperbolic sine.

Return value

The hyperbolic sine of the angle *num*.

Remarks

Uses FLOAT parameter and return for use in time critical applications.

See Also: [SINH](#), [SINH D](#), [FSINH D](#)

Example usage

```
PRINT FSINH(1.223)
```

12.178FSINH D

Syntax

FLOAT = FSINH D(num as FLOAT)

Description

Calculates the hyperbolic sine of an angle.

Parameters

num - The angle in degrees.

Return value

The hyperbolic sine of the angle *num*

Remarks

Uses FLOAT parameter and return for time critical applications.

See Also: [SINH](#), [SINH D](#), [FSINH](#)

Example usage

```
PRINT FSINH D(45.0)
```

12.179FTAN

Syntax

FLOAT = FTAN(num as FLOAT)

Description

Calculates the tangent of an angle.

Parameters

num - The angle in radians.

Return value

The tangent of the angle *num*.

Remarks

Uses FLOAT parameter and return for time critical applications.

See Also [TAN](#), [TAN D](#), [FTAN D](#)

Example usage

```
PRINT FTAN(2)
```

12.180FTAND

Syntax

FLOAT = FTAND(num as FLOAT)

Description

Calculates the tangent of an angle.

Parameters

num - Angle in degrees.

Return value

The tangent of the angle *num*.

Remarks

Uses FLOAT parameter and return for time critical applications.

See Also: [TAN](#), [TAND](#), [FTAN](#)

Example usage

```
PRINT FTAND(46.0)
```

12.181 FTANH

Syntax

FLOAT = FTANH(num as FLOAT)

Description

Calculates the hyperbolic tangent of an angle.

Parameters

num - Angle in radians.

Return value

The hyperbolic tangent of the angle *num*.

Remarks

Uses FLOAT parameter and return for time critical applications.

See Also: [TANH](#), [TANH D](#), [FTANH D](#)

Example usage

```
PRINT FTANH(1.113)
```

12.182 FTANH D

Syntax

FLOAT = FTANH D(num as FLOAT)

Description

Calculates the hyperbolic tangent of a number

Parameters

num - Angle in degrees

Return value

The hyperbolic tangent of the angle *num*.

Remarks

Uses FLOAT parameter and return for time critical applications.

See Also: [TANH](#), [TANHD](#), [FTANH](#)

Example usage

```
PRINT FTANHD(46.1)
```

12.183GET

Syntax

GET(*vFile* as BFILE,*record* as INT,*var* as POINTER)

Description

Gets one record from the random access binary file *vFile* and stores the result in the variable *var*. *record* must be greater than zero and the file must have been opened by OPENFILE. *var* can be any built-in or user defined variable type. Use EOF to determine when end of file has been reached.

Parameters

vFile - A binary file opened with the OPENFILE function.

record - The ones based record number.

var - Any built in or user defined type.

Return value

None

Remarks

See Also: [OPENFILE](#), [PUT](#)

Example usage

```
GET myfile, 20, phone_data
```

12.184GETBITMAPSIZE

Syntax

GETBITMAPSIZE(*handle* as UINT,*vWidth* as UINT BYREF,*vHeight* as UINT BYREF)

Description

Returns the width and height of a bitmap loaded by LOADIMAGE

Parameters

handle - [in] Handle returned by LOADIMAGE of type @IMGBITMAP

vWidth - [out] UINT variable to receive the width

vHeight - [out] UINT variable to receive the height

Return value

None

Remarks

See Also: [LOADIMAGE](#)

Example usage

```
DEF w,h as UINT
GETBITMAPSIZE myimage,w,h
```

12.185GETCAPTION

Syntax

STRING = GETCAPTION(win as WINDOW)

Description

Returns the caption text of a window or dialog

Parameters

win - Window or dialog to retrieve the caption text.

Return value

A string containing the text

Remarks

See Also: [SETCAPTION](#)

Example usage

```
cap = GETCAPTION(mywnd)
```

12.186GETCARETPOSITION

Syntax

INT = GETCARETPOSITION(win as WINDOW,x as INT BYREF,y as INT BYREF)

Description

Retrieves the current caret position in the window.

Parameters

win - WINDOW containing the caret.

x, y - The current caret position is stored in these variables.

Return value

TRUE if the caret exists and is being displayed in a window. FALSE otherwise.

Remarks

The *win* parameter is not used by the function and can be passed an uninitialized WINDOW variable. The variables *x* and *y* must be of type INT. The coordinates returned will always be in

reference to the client area of the window containing the caret.

Example usage

```
DEF x,y AS INT
GETCARETPOSITION mywin, x, y
```

12.187GETCLIENTSIZE

Syntax

GETCLIENTSIZE(win as WINDOW, l as UINT BYREF, t as UINT BYREF, w as UINT BYREF, h as UINT BYREF)

Description

Returns the size of the client area of the window or dialog.

Parameters

win - Window or dialog.

l, t, w, h - Variables of type UINT to receive the left, top, width and height of the client area.

Return value

None

Remarks

The left and top will always be 0 with this function.

See Also: [GETSIZE](#), [GETSCREENSIZE](#)

Example usage

```
DEF l,t,w,h as UINT
GETCLIENTSIZE mywnd, l, t, w, h
```

12.188GETCOMPROPERTY

Syntax

INT = GetComProperty(IDispatch obj, string fmt, uint value, string property, ...)

Description

Retrieves the property of a COM object using script syntax.

Parameters

obj - The Object returned by the CREATECOMOBJECT command.

fmt - Formatting string for the returned property.

value - Address of a variable to store the property in.

property - Name of the property to retrieve.

... - Optional parameter list.

Return value

0 for Success.

Remarks

Pass the address of your variable using the & operator. Parameters can be supplied to property names when needed.

Formatting strings are specified by using C printf-like specifiers. The table below shows the supported specifiers:

Identifier	Type
%d	INT
%u	UINT
%e	DOUBLE
%b	INT
%v	VARIANT UDT.
%B	BSTR - Created with the AllocSysString API function
%s	STRING
%S	WSTRING
%T	WSTRING
%o	IDispatch COM object
%O	IUnknown COM object
%t	C time_t UDT
%W	SYSTEMTIME UDT
%f	FILETIME UDT
%D	C date type.
%p	POINTER
%m	Specifies a missing/optional argument

Example usage(s)

```
GetComProperty(_xmlDoc, "%s", &pTemp, ".documentElement.Attributes.getNamedItem(%s).v
```

```
IDispatch Connection
POINTER szResponse
INT _status
Connection = CreateComObject("MSXML2.XMLHTTP.3.0", "")
CallObjectMethod(Connection, ".Open(%s, %s, %b)", "GET", "http://myserver.com/test.xml")
CallObjectMethod(Connection, ".Send")
GetComProperty(Connection, "%d", &_status, ".status")
if _status = 200
    GetComProperty(Connection, "%T", &_szResponse, ".ResponseXML.xml")
    PRINT w2s(*<wstring>_szResponse)
    FreeComString(_szResponse)
endif
Connection->Release()
```

12.189 GETCONTROLHANDLE

Syntax

handle = GETCONTROLHANDLE(win as WINDOW, id as INT)

Description

Returns the windows handle (HWND) of a control in a dialog or window..

Parameters

win - Window or dialog containing control.

id - Identifier of the control.

Return value

The windows handle of the control.

Remarks

Controls located in a dialog don't exist until DOMODAL or SHOWDIALOG is called. To retrieve the handle of a control located in a dialog use GETCONTROLHANDLE in response to the @IDINITDIALOG message

See Also: [CONTROL](#), [CONTROLEX](#)

Example usage

```
hProgress = GETCONTROLHANDLE(d1, 20)
```

12.190 GETCONTROLTEXT

Syntax

STRING = GETCONTROLTEXT(win as WINDOW, id as UINT)

Description

Retrieves the text of a control. Equivalent to the WM_GETTEXT windows message.

Parameters

win - Window or dialog containing the control

id - Identifier of the control

Return value

A string containing the controls text.

Remarks

For edit controls GETCONTROLTEXT will retrieve the text in the edit control. For all other controls it will retrieve the caption text of the control.

See Also: [SETCONTROLTEXT](#)

Example usage

```
ed$ = GETCONTROLTEXT(mydlg, 1)
```

12.191 GETDATA

Syntax

GETDATA(blockname as DATABLOCK BYREF, var as ANYTYPE)

Description

Reads a datum from the indicated data block and increments the data pointer.

Parameters

blockname - Name of the data block specified in the DATABEGIN statement

var - Variable to store the datum into.

Return value

None

Remarks

It is possible to read past the end of data so a count should be kept or the last datum used as a terminator. Currently IWBASIC supports INT, FLOAT, DOUBLE and STRING data items. GETDATA will convert between most common variable types. Reset the data pointer for the block by using RESTORE.

See Also: [DATABEGIN](#), [DATAEND](#), [DATA](#), [RESTORE](#)

Example usage

```
DEF a as INT

RESTORE mydata

FOR x = 1 to 6
    GETDATA mydata, a
    PRINT a
NEXT x

DO:UNTIL INKEY$ <> ""
END

DATABEGIN mydata
DATA 1, 2, 3
DATA 4, 10, 2002
DATAEND
```

12.192 GETDEFAULTPRINTER

Syntax

STRING = GETDEFAULTPRINTER

Description

Returns the name of the default printer, if any, assigned to the system. The name can subsequently be used in the OPENPRINTER function.

Parameters

None

Return value

A string containing the printer name or an empty string if no printing devices are set as default.

Remarks

See Also: [OPENPRINTER](#)

Example usage

```
printer = GETDEFAULTPRINTER
```

12.193 GETEXCEPTIONCODE

Syntax

GETEXCEPTIONCODE

Description

Used to create a user generated exception.

Parameters

value - Can be a variable, a literal number, or a literal string.

Return value

None

Remarks

See Also: [TRY](#), [ENDTRY](#), [CATCH](#), [ENDCATCH](#), [Exception Handling](#)

Example usage

See [Exception Handling](#)

12.194 GETEXCEPTIONINFORMATION

Syntax

GETEXCEPTIONINFORMATION

Description

Used to create a user generated exception.

Parameters

value - Can be a variable, a literal number, or a literal string.

Return value

None

Remarks

See Also: [TRY](#), [ENDTRY](#), [CATCH](#), [ENDCATCH](#), [Exception Handling](#)

Example usage

See [Exception Handling](#)

12.195 GETFOLDERPATH

Syntax

```
STRING = GETFOLDERPATH(csidl as INT)
```

Description

Returns the path of a folder on the system.

Parameters

csidl - An integer specifying the folder to retrieve.

Return value

A string containing the path to the folder. The path returned does not contain a trailing slash.

Remarks

CSIDL values predefined are:

```
@CSIDL_DESKTOP  
@CSIDL_INTERNET  
@CSIDL_PROGRAMS  
@CSIDL_CONTROLS  
@CSIDL_PRINTERS  
@CSIDL_PERSONAL  
@CSIDL_FAVORITES  
@CSIDL_STARTUP  
@CSIDL_RECENT  
@CSIDL_SENDTO  
@CSIDL_BITBUCKET  
@CSIDL_STARTMENU  
@CSIDL_MYDOCUMENTS  
@CSIDL_MYMUSIC  
@CSIDL_MYVIDEO
```

@CSIDL_DESKTOPDIRECTORY
@CSIDL_DRIVES
@CSIDL_NETWORK
@CSIDL_NETHOOD
@CSIDL_FONTS
@CSIDL_TEMPLATES
@CSIDL_COMMON_STARTMENU
@CSIDL_COMMON_PROGRAMS
@CSIDL_COMMON_STARTUP
@CSIDL_COMMON_DESKTOPDIRECTORY
@CSIDL_APPDATA
@CSIDL_PRINTHOOD
@CSIDL_LOCAL_APPDATA
@CSIDL_ALTSTARTUP
@CSIDL_COMMON_ALTSTARTUP
@CSIDL_COMMON_FAVORITES
@CSIDL_INTERNET_CACHE
@CSIDL_COOKIES
@CSIDL_HISTORY
@CSIDL_COMMON_APPDATA
@CSIDL_WINDOWS
@CSIDL_SYSTEM
@CSIDL_PROGRAM_FILES
@CSIDL_MYPICTURES
@CSIDL_PROFILE
@CSIDL_SYSTEMX86
@CSIDL_PROGRAM_FILESX86
@CSIDL_PROGRAM_FILES_COMMON
@CSIDL_PROGRAM_FILES_COMMONX86
@CSIDL_COMMON_TEMPLATES
@CSIDL_COMMON_DOCUMENTS
@CSIDL_COMMON_ADMINTOOLS
@CSIDL_ADMINTOOLS
@CSIDL_CONNECTIONS
@CSIDL_COMMON_MUSIC
@CSIDL_COMMON_PICTURES
@CSIDL_COMMON_VIDEO
@CSIDL_RESOURCES
@CSIDL_RESOURCES_LOCALIZED
@CSIDL_COMMON_OEM_LINKS
@CSIDL_CDBURN_AREA
@CSIDL_COMPUTERSNEARME

Example usage

```
PRINT GetFolderPath(@CSIDL_COMMON_DOCUMENTS)
```

12.196 GETHDC

Syntax

```
UINT = GETHDC(win as WINDOW)
```

Description

Safe way to access a device context to an IW BASIC created window. The returned dc can be used with any Windows API function that requires a device context to the window.

Parameters

win - Window to obtain the drawing context.

Return value

A handle to a device context or NULL if the device context could not be obtained.

Remarks

A dc may also be obtained for caching purposes. While you hold the dc all drawing functions will use the cached copy instead of obtaining a new one for each operation which can significantly speed up drawing operations. The device context returned is compatible with IW BASIC's autodraw windows and will be preset with the current window attributes such as font, pen colors and drawing modes.

The dc must be returned to the system by using `RELEASEHDC` when you are finished using it. Every call to `GETHDC` must be matched in pairs to `RELEASEHDC`.

See Also: [RELEASEHDC](#)

Example usage

```
hdc = GETHDC(win)
...
RELEASEHDC(win, hdc)
```

12.197 GETKEYSTATE

Syntax

```
INT = GETKEYSTATE(num as INT)
```

Description

Returns the asynchronous status of the specified virtual key

Parameters

num - Virtual key code to test

Return value

Return is greater than 0 if the key is currently being held down.

Remarks

On 95/98/ME will not work in a console executable unless there is a message processing loop.
See Also: Appendix

Example usage

```
IF GETKEYSTATE(0x1B) THEN PRINT "ESC pressed"
```

12.198 GETPIXEL

Syntax

UINT = GETPIXEL(win as WINDOW,x as INT,y as INT)

Description

Returns the color of a pixel at the specified coordinates.

Parameters

win - Window to get pixel color.

x, y - coordinate of pixel

Return value

The RGB color of the pixel at coordinate *x,y*

Remarks

See Also: [PSET](#)

Example usage

```
IF GETPIXEL(mywnd,100,2) = RGB(255,0,0) THEN PRINT "The pixel is red"
```

12.199 GETPOSITION

Syntax

GETPOSITION(win as WINDOW,x as INT BYREF,y as INT BYREF)

Description

Returns the current drawing position in the window. Also known as the pen position.

Parameters

win - Window to retrieve drawing position from.

x, y - The position is stored in these variables.

Return value

None

Remarks

The variables *x* and *y* must be of type INT. The current drawing position is set with the MOVE statement, graphics operations, and text PRINTing.

See Also: [MOVE](#)

Example usage

```
DEF x,y as INT
PRINT myWin, "Some text to print"
'Get the position at the end of the last printed line
GETPOSITION myWin, x, y
```

12.200 GetProgressPosition

Syntax

INT = GetProgressPosition(win as WINDOW, id as UINT)

Description

Retrieves the current position of the progress bar.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

Returns an INT value that represents the current position of the progress bar.

Remarks

None.

Example usage

```
IF GetProgressPosition(cp, IDPROGRESSBAR) > 500 THEN SetProgressPosition(cp, IDPROGRESSB
```

12.201 GETRESOURCELENGTH

Syntax

INT = GETRESOURCELENGTH(identifier, type)

Description

Returns the length of the resource that is stored in the executable

Parameters

identifier - Resource name or numeric identifier.

type - Type name or numeric identifier.

Return value

The length of the resource in bytes

Remarks

See Also: [LOADRESOURCE](#)

The length of standard windows resource types can be found by specifying one of the following for the *type* parameter:

@RESCURSOR

@RESBITMAP

@RESICON

@RESMENU

@RESDIALOG

@RESSTRING

@RESACCEL

@RESDATA

@RESMESSAGETABLE

@RESGROUPCURSOR

@RESGROUPICON

@RESVERSION

Example usage

```
PRINT GETRESOURCELENGTH("mydata",@RESDATA)
```

12.202 GETSCREENSIZE

Syntax

GETSCREENSIZE(*w* as UINT BYREF,*h* as UINT BYREF)

Description

.Returns the current screen size

Parameters

w - [out] The width of the screen.

h - [out] The height of the screen.

Return value

None

Remarks

The *h* and *w* parameters must be defined as type UINT.

See Also: [GETCLIENTSIZE](#), [GETSIZE](#)

Example usage

```
DEF screenH,screenW as UINT  
GETSCREENSIZE screenW,screenH
```

12.203 GETSCROLLPOS

Syntax

INT = GETSCROLLPOS(win as WINDOW, id as INT)

Description

Gets the current scroll position of a scrollbar control or the windows scrollbars

Parameters

win - Window or dialog containing the scrollbar to query.

id - Identifier of a control or windows scrollbar.

Return value

The current position of the scrollbar

Remarks

Use and *id* of -1 for the windows horizontal scrollbar, -2 for the windows vertical scrollbar, or any other value for a scroll bar control.

See Also: [SETSCROLLPOS](#)

Example usage

```
hPos = GETSCROLLPOS(win, -1)
```

12.204 GETSCROLLRANGE

Syntax

GETSCROLLRANGE(win as WINDOW, id as INT, vMin as INT BYREF, vMax as INT BYREF)

Description

Retrieves the scroll range of a scrollbar.

Parameters

win - [in] Window or dialog containing the scrollbar to query.

id - [in] Identifier of the scrollbar control or windows scrollbar

vMin - [out] Variable to receive the minimum range value.

vMax - [out] Variable to receive the maximum range value.

Return value

None

Remarks

Use and *id* of -1 for the windows horizontal scrollbar, -2 for the windows vertical scrollbar, or any other value for a scroll bar control.

See Also: [SETSCROLLRANGE](#)

Example usage

```
DEF min,max as INT
GETSCROLLRANGE mydlg, 5, min, max
```

12.205 GETSELECTED

Syntax

INT = GETSELECTED(win as WINDOW, id as INT)

Description

Returns the zero-based index of the currently selected item in the list box of a combo or *single selection* list box.

Parameters

win - Window or dialog containing the control.

id - Identifier of the combo box or list box.

Return value

The zero based index of the currently selected item.

Remarks

See Also: [SETSELECTED](#)

Example usage

```
pos = GETSELECTED(mydlg, 5)
```

12.206 GETSIZE

Syntax

GETSIZE(win as WINDOW, l as INT BYREF, t as INT BYREF, w as INT BYREF, h as INT BYREF, OPT id=0 as UINT)

Description

Gets the size of a window, dialog or control. The dimensions returned include the borders and caption.

Parameters

win - [in] Window or dialog

l, *t*, *w*, *h* - [out] Variables to receive the dimensions

id - [in] Optional control identifier.

Return value

None

Remarks

Variables must be of type INT. Coordinates are returned relative to the screen.

See Also: [SETSIZE](#)

Example usage

```
DEF l,t,w,h as INT
GETSIZE mywindow, l, t, w, h
```

12.207 GetSpinnerBase

Syntax

INT = GetSpinnerBase(win as WINDOW, id as UINT)

Description

Retrieves the current radix base for the spin button control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

Returns the current radix base for the control. Either base 10 or 16.

Remarks

The default is base 10.

Example usage

```
base = GetSpinnerBase(demo, 300)
```

12.208 GetSpinnerBuddy

Syntax

UINT = GetSpinnerBuddy(win as WINDOW, id as UINT)

Description

Returns the id to the current buddy control for the spin button control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the spinner control.

Return value

ID the current buddy control, or 0 if the buddy has not been set.

Remarks

None

Example usage

```
buddy = GetSpinnerBuddy(demo, 300)
```

12.209 GetSpinnerPosition

Syntax

INT = GetSpinnerPosition(win as WINDOW, id as UINT)

Description

Returns the position of the spin button control

Parameters

win - Dialog or window containing the control.

id - Identifier of the spinner control.

Return value

The current position of the the spin button control.

Remarks

When this method is called, the spin button control updates its current position based on the caption of the buddy window. If there is no buddy window or if the caption specifies an invalid or out-of-range value then the return value is undefined.

Example usage

```
pos = GetSpinnerPos(demo, 300)
```

12.210 GetSpinnerRangeMax

Syntax

INT = GetSpinnerRangeMax(win as WINDOW, id as UINT)

Description

Returns the upper limit of the range of the spinner control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the spinner control.

Return value

The upper limit set by [SetSpinnerRange](#)

Remarks

None

Example usage

```
max = GetSpinnerRangeMax(demo, 300)
```

12.211 GetSpinnerRangeMin

Syntax

INT = GetSpinnerRangeMin(win as WINDOW, id as UINT)

Description

Returns the lower limit of the range of the spinner control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the spinner control.

Return value

The lower limit set by [SetSpinnerRange](#)

Remarks

None

Example usage

```
max = GetSpinnerRangeMax(demo, 300)
```

12.212 GETSTARTPATH

Syntax

STRING = GETSTARTPATH

Description

Returns the path to your executable.

Parameters

None

Return value

The full path to the directory containing the running executable.

Remarks

Path includes a trailing slash.

Example usage

```
filename = GETSTARTPATH + "bug.bmp"
```

12.213 GETSTARTPATHW

Syntax

WSTRING = GETSTARTPATHW

Description

Returns the path to your executable.

Parameters

None

Return value

The full path to the directory containing the running executable.

Remarks

Path includes a trailing slash.

Example usage

```
filename = GETSTARTPATHW + L"bug.bmp"
```

12.214 GETSTATE

Syntax

INT = GETSTATE(win as WINDOW, id as INT)

Description

Returns the state of a checkbox or radio button control.

Parameters

win - Window or dialog containing the control.

id - Controls identifier.

Return value

Returns 1 if control is checked and 0 if control is unchecked.

Remarks

Radio buttons in a group are mutually exclusive. When a radio button is selected your program will receive an @IDCONTROL message with @CONTROLID containing the newly selected radio button. If the radio button is created outside of a group, you will need to use GETSTATE to determine whether the button is selected.

See Also: [SETSTATE](#)

Example usage

```
state = GETSTATE(mydlg, 5)
```

12.215 GETSTRING

Syntax

STRING = GETSTRING(win as WINDOW, id as INT, pos as UINT)

Description

Returns a string in a list box or combo box. *pos* is a zero based index.

Parameters

win - Window or dialog containing control.

id - Identifier of the list box or combo box control.

pos - String position to return.

Return value

The string located at *pos*

Remarks

See Also: [ADDSTRING](#), [INSERTSTRING](#), [DELETESTRING](#)

Example usage

```
strSel = GETSTRING(mydlg, 7, 0)
```

12.216 GETSTRINGCOUNT

Syntax

INT = GETSTRINGCOUNT(win as WINDOW, id as INT)

Description

Returns the number of strings in a list box or combo box control.

Parameters

win - Window or dialog containing the control.

id - Identifier of the list box or combo box control

Return value

The number of strings in the control.

Remarks

None.

Example usage

```
count = GETSTRINGCOUNT(dlg1, 9)
```

12.217 GETTEXTSIZE

Syntax

GETTEXTSIZE(*win* as WINDOW,*text* as STRING,*vWidth* as INT BYREF,*vHeight* as INT BYREF)

Description

Retrieves the size of a string in pixels when printed to the window.

Parameters

win - [in] Window for test.

text - [in] String to measure.

vWidth, *vHeight* - [out] Variables to receive text size.

Return value

None

Remarks

vWidth and *vHeight* must be variables of type INT. The size is calculated using the current font of the window and is useful for determining line positions.

Example usage

```
DEF tWidth, tHeight as INT
GETTEXTSIZE win1, "Tell me", tWidth, tHeight
```

12.218 GETTHUMBPOS

Syntax

INT = GETTHUMBPOS(*win* as WINDOW,*id* as INT)

Description

Gets the current thumb track position of a scrollbar control or a windows scrollbars.

Parameters

win - Window or dialog containing the scrollbar to query.

id - Identifier of a control or windows scrollbar.

Return value

The current thumb track position of the scrollbar

Remarks

Use and *id* of -1 for the windows horizontal scrollbar, -2 for the windows vertical scrollbar, or any other value for a scroll bar control. This function returns the full 32 bit position of the thumb track box. Use in response to the @SBTHUMBTRACK code or the @SBTHUMBPOS code while processing an @IDHSCROLL or @IDVSCROLL message.

See Also: [GETSCROLLPOS](#)

Example usage

```
hPos = GETTHUMBPOS(win, -1)
```

12.219 GetTrackBarLineSize

Syntax

```
INT = GetTrackBarLineSize(win as WINDOW, id as UINT)
```

Description

Retrieves the number of logical positions the TrackBar's slider moves in response to keyboard input from the arrow keys.

Parameters

win - Dialog or window containing the control.

id - Identifier of the TrackBar control.

Return value

Returns a 32-bit value that specifies the line size for the trackbar.

Remarks

Retrieves the number of logical positions the trackbar's slider moves in response to keyboard input from the arrow keys, such as the RIGHT ARROW or DOWN ARROW keys. The logical positions are the integer increments in the trackbar's range of minimum to maximum slider positions.

Example usage

```
sz = GetTrackBarLineSize(demo, 300)
```

12.220 GetTrackBarPageSize

Syntax

```
INT = GetTrackBarPageSize(win as WINDOW, id as UINT)
```

Description

Retrieves the number of logical positions the trackbar's slider moves in response to keyboard input.

Parameters

win - Dialog or window containing the control.

id - Identifier of the TrackBar control.

Return value

Returns a 32-bit value that specifies the page size for the trackbar.

Remarks

Retrieves the number of logical positions the trackbar's slider moves in response to keyboard input, such as the PAGE UP or PAGE DOWN keys, or mouse input, such as clicks in the

trackbar's channel. The logical positions are the integer increments in the trackbar's range of minimum to maximum slider positions

Example usage

```
sz = GetTrackBarPageSize(demo, 300)
```

12.221 GetTrackBarPosition

Syntax

INT = GetTrackBarPosition(win as WINDOW, id as UINT)

Description

Retrieves the current logical position of the slider in a trackbar.

Parameters

win - Dialog or window containing the control.

id - Identifier of the TrackBar control.

Return value

Returns a 32-bit value that specifies the current logical position of the trackbar's slider.

Remarks

The logical positions are the integer values in the trackbar's range of minimum to maximum slider positions.

Example usage

```
pos = GetTrackBarPosition(demo, 300)
```

12.222 GetTrackBarRangeMax

Syntax

INT = GetTrackBarRangeMax(win as WINDOW, id as UINT)

Description

Returns the upper limit of the range of the trackbar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the TrackBar control.

Return value

The upper limit set by [SetTrackbarRange](#).

Remarks

None

Example usage

```
max = GetTrackBarRangeMax(demo, 300)
```

12.223 GetTrackBarRangeMin

Syntax

INT = GetTrackBarRangeMin(win as WINDOW, id as UINT)

Description

.Returns the lower limit of the range of the trackbar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the TrackBar control.

Return value

The lower limit set by [SetTrackbarRange](#).

Remarks

None

Example usage

```
min = GetTrackBarRangeMin(demo, 300)
```

12.224 GLOBAL

Syntax

GLOBAL function_name
GLOBAL variable_name,...

Description

Declares a subroutine or variable as being globally visible.

Parameters

function_name - The name of the subroutine to make global.

variable_name - The name(s) of a variable(s) to make global.

Return value

None

Remarks

Subroutines and variables are normally private to the source file they are defined/declared in. Declaring variables as GLOBAL allows other source file modules to use them when using the EXTERN keyword. Declaring subroutines as GLOBAL allows other source file modules to use them when using a DECLARE EXTERN statement. EXTERN and GLOBAL are the heart of

multi-module programming.

See Also [EXTERN](#) , [DECLARE](#), [PROJECTGLOBAL](#)

Example usage

```
'in the defining source file
GLOBAL myvariable
GLOBAL myfunction
DEF myvariable as UINT

SUB myfunction(a as INT),UINT
    RETURN a+5
ENDSUB
-----
'in another source file
EXTERN myvariable as UINT
DECLARE EXTERN myfunction(a as INT),UINT
myvariable = myfunction(7)
```

12.225GOSUB

Syntax

```
GOSUB name
GOSUB name(parameters)
```

Description

Calls a subroutine. Included for backwards compatibility with other languages.

Parameters

name - Name of subroutine to call
parameters - Comma separated list of parameters

Return value

None

Remarks

It is not necessary to use GOSUB as you can call any subroutine directly by its name.

See Also: [SUB](#)

Example usage

```
GOSUB mysub (10, 22)
```

12.226GOTO

Syntax

```
GOTO label_name
```

Description

Causes a jump to a label in your code.

Parameters

label_name - The name in the LABEL statement.

Return value

None

Remarks

Can not be used to jump into or out of a subroutine.

See Also: [LABEL](#)

Example usage

```
FOR x = 1 to 100
    IF x = 50 THEN GOTO fexit
NEXT x

LABEL fexit
PRINT x
```

12.227HeaderControl

Syntax

UINT = HeaderControl(win as WINDOW,l as INT,t as INT,w as INT,h as INT,flags as INT,exStyle as INT,id as UINT)

Description

Creates a header control.

Parameters

win - Parent window or dialog containing the control.

l, t, w, h - Position and dimension of the control.

flags - Styles for the control.

exStyle - Extended window styles for the control

id - Control identifier.

Return value

If used with a window the return value is the handle of the control. If used with a dialog the return value is 0 and the handle to the control can be obtained with GETCONTROLHANDLE during the @IDINITDIALOG message

Remarks

The control can also be created using CONTROLEX with a window class of "SysHeader32".

Styles

@HDS_HORZ - Creates a header control with a horizontal orientation.

@HDS_BUTTONS - Each item in the control looks and behaves like a push button.
@HDS_HOTTRACK - Enables hot tracking.
@HDS_HIDDEN - Indicates a header control that is intended to be hidden.
@HDS_DRAGDROP - Allows drag-and-drop reordering of header items.
@HDS_FULLDRAG - Causes the header control to display column contents even while the user resizes a column.

Example usage

```
HeaderControl cp,0,0,640,20,@HDS_BUTTONS,0,IDHEADER
```

12.228HeapClear

Syntax

HeapClear

Description

Clears the internal heap stack. Used internally by string functions.

Parameters

None

Return value

None

Remarks

Only for use by command implementers. Do not call this function from high level code as severe memory corruption can occur.

See Also: [PushHeap](#)

Example usage

None

12.229HEX\$ / WHEX\$

Syntax

STRING = HEX\$(num as UINT64)

WSTRING = WHEX\$(num as UINT64)

Description

Converts a number to a string representing the hexadecimal notation of that number.

Parameters

num - Number to convert

Return value

A string with the raw hexadecimal notation.

Remarks

HEX\$ can handle 64 bit numbers, the maximum string returned will be 16 digits long. Negative quantities are represented by their twos complement in hexadecimal.

See Also: [STR\\$](#), [VAL](#)

Example usage

```
PRINT "0x" + HEX$(65534)
```

12.230hcDeleteItem

Syntax

hcDeleteItem(win as WINDOW,id as UINT,index as INT)

Description

Deletes an item from a header control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero based index of the item to delete.

Return value

None

Remarks

None.

Example usage

```
hcDeleteItem demo,400,1
```

12.231hcGetItemCount

Syntax

INT = hcGetItemCount(win as WINDOW,id as UINT)

Description

Retrieves a count of the items in a header control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

The count of items or -1 if unsuccessful.

Remarks

None

Example usage

```
count = hcGetItemCount(demo, 400)
```

12.232hcGetItemData

Syntax

UINT = hcGetItemData(win as WINDOW, id as UINT, index as INT)

Description

Returns the application-defined item data associated with an item in the header control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero based index.

Return value

Application-defined item data.

Remarks

The data is a 32 bit value set with [hcSetItemData](#)

Example usage

```
field = hcGetItemData(win, 400, 3)
```

12.233hcGetItemRect

Syntax

WINRECT = hcGetItemRect(win as WINDOW, id as UINT, index as INT)

Description

Retrieves the bounding rectangle for a specified item in a header control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero based index.

Return value

A WINRECT UDT containing the bounding rectangle.

Remarks

On error the rectangle will have dimensions of 0.

Example usage

```
WINRECT rc
rc = hcGetItemRect(demo, 400, 2)
```

12.234hcGetItemText

Syntax

STRING = hcGetItemText(win as WINDOW, id as UINT, index as INT)

Description

Returns the text of an item in the header control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero based index.

Return value

A string containing the text of the item.

Remarks

None

Example usage

```
text = hcGetItemText(demo, 300, 2)
```

12.235hcGetItemWidth

Syntax

INT = hcGetItemWidth(win as WINDOW, id as UINT, index as int)

Description

Returns the width, in pixels of an item in the header control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero based index.

Return value

The width of the item.

Remarks

None.

Example usage

```
width = hcGetItemWidth(demo,400,2)
```

12.236hcInsertItem

Syntax

hcInsertItem(win as WINDOW,id as UINT,index as INT,text as STRING,width as INT,OPT image=-2 as INT)

Description

Inserts an item into the header control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Index of the item after which the new item is to be inserted. The new item is inserted at the end of the header control if *index* is greater than or equal to the number of items in the control. If *index* is zero, the new item is inserted at the beginning of the header control.

text - Text of the item.

width - The width of the item.

image - Optional. The zero based index of an image in the image list.

Return value

None.

Remarks

None

Example usage

```
FOR x = 0 TO NUM_COLUMNS-1
    hcInsertItem cp, IDHEADER, x, "Column "+LTRIM$(STR$(x+1)), 100
    hcSetItemJustify cp, IDHEADER, x, HDF_RIGHT
NEXT x
```

12.237hcSetImageList

Syntax

hcSetImageList(win as WINDOW,id as UINT,himl as UINT)

Description

Sets the image list used by the header control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

himgl - Handle to an image list created with the ImageList_Create API function.

Return value

None

Remarks

None

Example usage

```
hcSetImageList demo,400,himagelist
```

12.238 hcSetItemData

Syntax

```
hcSetItemData(win as WINDOW,id as UINT,index as INT,value as UINT)
```

Description

Sets the application-defined item data associated with an item in the header control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero based index.

value - An application defined 32 bit value.

Return value

None

Remarks

The data can be retrieved with [hcGetItemData](#).

Example usage

```
hcSetItemData(win,400,3,0x4FF)
```

12.239 hcSetItemJustify

Syntax

```
hcSetItemJustify(win as WINDOW,id as UINT,index as INT,justify as INT)
```

Description

Sets the justification of the text of an item in the header control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero based index.

justify - Value indicating the justification.

Return value

None

Remarks

The *justify* value can be one of:

HDF_CENTER - The item's contents are centered.

HDF_LEFT - The item's contents are left-aligned.

HDF_RIGHT- The item's contents are right-aligned.

Example usage

```
FOR x = 0 TO NUM_COLUMNS-1
    hcInsertItem cp, IDHEADER, x, "Column "+LTRIM$(STR$(x+1)), 100
    hcSetItemJustify cp, IDHEADER, x, HDF_RIGHT
NEXT x
```

12.240hcSetItemText

Syntax

hcSetItemText(win as WINDOW, id as UINT, index as int, text as STRING)

Description

Changes the text of an item in the header control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero based index.

text - The text of the item.

Return value

None

Remarks

None

Example usage

```
hcSetItemText demo,400,2,"Column 3"
```

12.241hcSetItemWidth

Syntax

```
hcSetItemWidth(win as WINDOW,id as UINT,index as INT,width as INT)
```

Description

Sets the width,in pixels, of an item in the header control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero based index.

width - New width of the item.

Return value

None

Remarks

None

Example usage

```
hcSetItemWidth(demo,400,2,100)
```

12.242IF

Syntax

```
IF condition THEN action { ELSE action }
```

```
IF condition {THEN}  
  actions  
{ ELSE {IF condition}  
  actions }  
ENDIF
```

Description

Beginning of an IF statement or IF block.

Parameters

condition - Any condition that will be either TRUE or FALSE.

Return value

None

Remarks

If the *condition* is true then all of the statements following the IF statement will be executed up to and including the statement before an ELSE, ELSEIF or ENDIF statement. All IF statement blocks must end with a corresponding ENDIF statement unless using the single line format.

Example usage

```
IF a = 7 THEN PRINT "a=7"  
IF a = 6  
    PRINT "a=6"  
ELSE  
    PRINT "a <> 6"  
ENDIF
```

12.243IMPORT

Syntax

IMPORT

Description

Reserved word. Used in DECLARE statements.

Parameters

N/A

Return value

N/A

Remarks

See Also: [DECLARE](#)

Example usage

See [DECLARE](#)

12.244INKEY\$

Syntax

STRING = INKEY\$(OPT raw=0 as INT)

Description

Returns a single character from the keyboard in console mode

Parameters

raw - Optional. If equal to 1 the INKEY\$ function will return the raw virtual code of the next key pressed, including function and special keys.

Return value

A one character string containing the value of the key pressed.

Remarks

INKEY\$ is non blocking and will return an empty string, "", if there is no input present. OPENCONSOLE must have been executed prior to using the command or the process will lockup waiting for input.

See Also: [INPUT](#), [GETKEYSTATE](#), [OPENCONSOLE](#)

Example usage

```
OPENCONSOLE
PRINT "Press Q to quit"
DO: UNTIL INKEY$ = "Q"
CLOSECONSOLE
END
```

12.245INPUT

Syntax

INPUT OPT prompt as STRING,var as ANYTYPE

Description

Gets data from the console window.

Parameters

prompt - [in] Optional text to display on input line.

var - [out] Variable to store input.

Return value

None

Remarks

OPENCONSOLE must have been executed prior to using the INPUT statement. The optional string is a prompt. The statement will wait until the user has entered the proper data. Returns when the 'enter' key is pressed. Valid variable types are CHAR, WORD, INT, UINT, INT64, UINT64, FLOAT, DOUBLE, and STRING.

Numeric input is converted with VAL. Strings do not include the new line character. The maximum string input is 255 characters.

See Also: [VAL](#)

Example usage

```
DEF A$:string
INPUT "Enter Your Name",A$
PRINT "Hello ",A$
```

12.246INSERTMENU

Syntax

INSERTMENU(win as WINDOW,hmenu as UINT,pos as UINT)

Description

Low level menu function used internally by menu creation macros.

Parameters

win - Window

hmenu - handle to popup menu created with CREATEMENU

pos - Zero based position on menu bar for insertion

Return value

None

Remarks

See Also: [CREATEMENU](#), [APPENDMENU](#)

Example usage

```
INSERTMENU mywnd, mymenu, 0
```

12.247INSERTSTRING

Syntax

INSERTSTRING(win as WINDOW,id as INT,pos as UINT,text as STRING)

Description

Inserts a string into a combo or list box control.

Parameters

win - Window or dialog containing the control.

id - Identifier of the list box or combo box control

pos - Zero based position to insert the string

text - String to insert.

Return value

None

Remarks

All other strings are moved down by one position.

See Also: [ADDSTRING](#), [DELETESTRING](#)

Example usage

```
INSERTSTRING mywindow, 7, 0, "New text"
```

12.248 INSTR

Syntax

UINT = INSTR(target as STRING,search as STRING,OPT pos = 1 as UINT)
UINT = WINSTR(target as WSTRING,search as WSTRING,OPT pos = 1 as UINT)

Description

Finds the first occurrence of a search string in a target string.

Parameters

target - String to perform the search on.

search - String to search for.

pos - Optional ones based starting position to begin the search.

Return value

Returns the ones based position of the first occurrence of the *search* string in the *target* string.
Returns 0 if the *search* string could not be found.

Remarks

Parameters of INSTR are different from other languages.

Example usage

```
A$ = "A string to search for something"  
PRINT INSTR(A$, "search")
```

12.249 INT

Syntax

integer = INT(num)

Description

Converts a floating point value to an integer.

Parameters

num - A DOUBLE or FLOAT

Return value

If num is a DOUBLE then the return is an INT64 otherwise an INT.

Remarks

INT truncates the decimal portion. This behavior is different than assigning a floating point value to an integer variable where the rounding mode will return the largest integer.

Example usage

```
PRINT INT(1.56)
```

12.250INT64

Syntax

INT64 = INT64(num)

Description

Converts a floating point value to an INT64.

Parameters

num - A numeric expression

Return value

An INT64 value.

Remarks

Example usage

```
PRINT INT64(1.56)
```

12.251INTERFACE

Syntax

INTERFACE name

Description

Begins defining a COM interface.

Parameters

name - Name of the interface. This will be used in the SET_INTERFACE command

Return value

None

Remarks

Each INTERFACE statement must be paired with an ENDINTERFACE. The only statement allowed between the two is STDMETHOD

See Also: [ENDINTERFACE](#), [SET INTERFACE](#), [STDMETHOD](#), [DEFINE GUID](#)

Example usage

```
INTERFACE IUnknown
    STDMETHOD QueryInterface(riid as POINTER, ppvObj as POINTER),INT
    STDMETHOD AddRef(),INT
    STDMETHOD Release(),INT
ENDINTERFACE
```

12.252IPClearAddress

Syntax

IPClearAddress(win as WINDOW,id as UINT)

Description

Clears the contents of the IP address control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

None

Remarks

None

Example usage

```
IPClearAddress demo,500
```

12.253IPControl

Syntax

UINT = IPControl(win as WINDOW,l as INT,t as INT,w as INT,h as INT,flags as INT,exStyle as INT,id as UINT)

Description

Creates an IP address control.

Parameters

win - Parent window or dialog containing the control.

l, t, w, h - Position and dimension of the control.

flags - Styles for the control.

exStyle - Extended window styles for the control

id - Control identifier.

Return value

If used with a window the return value is the handle of the control. If used with a dialog the return value is 0 and the handle to the control can be obtained with GETCONTROLHANDLE during the @IDINITDIALOG message

Remarks

The control can also be created using CONTROLEX with a window class of "SysIPAddress32".

Example usage

```
IPControl cp,40,80,206,20,@TABSTOP,0,IDIPADDRESS
ipSetAddress cp,IDIPADDRESS,127,0,0,1
```

12.254IPGetAddress

Syntax

IPGetAddress(win as WINDOW,id as UINT,f1 as INT BYREF,f2 as INT BYREF,f3 as INT BYREF,f4 as INT BYREF)

Description

Returns the IP address contained in the control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

f1, f2, f3, f4 - IP Address Fields

Return value

None.

Remarks

None

Example usage

```
INT f1, f2, f3, f4
IPGetAddress demo,500,f1,f2,f3,f4
PRINT "The IP Address is: ",f1,".",f2,".",f3,".",f4
```

12.255IPGetAddressDword

Syntax

UINT = IPGetAddressDword(win as WINDOW,id as UINT)

Description

Gets the address values for all four fields in the IP address control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

A UINT value that receives the address. The field 3 value will be contained in bits 0 through 7. The field 2 value will be contained in bits 8 through 15. The field 1 value will be contained in bits 16 through 23. The field 0 value will be contained in bits 24 through 31.

Remarks

None

Example usage

```
address = IPGetAddressDword(win, id)
f1 = (address >> 24) & 0xFF
f2 = (address >> 16) & 0xFF
f3 = (address >> 8) & 0xFF
f4 = address & 0xFF
```

12.256 IPIsBlank

Syntax

INT = IPIsBlank(win as WINDOW, id as UINT)

Description

Determines if all fields in the IP address control are blank.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

TRUE if all four fields are blank, FALSE otherwise.

Remarks

None

Example usage

```
IF IPIsBlank(demo, 500)
    IPSetAddress(demo, 500, 127, 0, 0, 1)
ENDIF
```

12.257 IPSetAddress

Syntax

IPSetAddress(win as WINDOW, id as UINT, f1 as INT, f2 as INT, f3 as INT, f4 as INT)

Description

Sets the address values for all four fields in the IP address control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

f1, f2, f3, f4 - IP Address Fields

Return value

None

Remarks

None

Example usage

```
IPControl cp,40,80,206,20,@TABSTOP,0,IDIPADDRESS  
ipSetAddress cp,IDIPADDRESS,127,0,0,1
```

12.258IPSetAddressDword

Syntax

IPSetAddressDword(win as WINDOW,id as UINT,address as UINT)

Description

Sets the address values for all four fields in the IP address control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

address - The IP address.

Return value

None.

Remarks

The address is a UINT value that contains the all four fields. The field 3 value is contained in bits 0 through 7. The field 2 value is contained in bits 8 through 15. The field 1 value is contained in bits 16 through 23. The field 0 value is contained in bits 24 through 31.

Example usage

```
IPSetAddressDword(demo, 500, address)
```

12.259IPSetRange

Syntax

IPSetRange(win as WINDOW,id as UINT,field as INT,low as CHAR,high as CHAR)

Description

Sets the valid range for the specified field in the IP address control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

field - Zero-based field index to which the range will be applied.

low - Lowest allowable value.

high - Highest allowable value.

Return value

None

Remarks

The allowable range is from 0 to 255 inclusive.

Example usage

```
IPSetRange demo, 500, 0, 10, 192
```

12.260 ISREF

Syntax

ISREF variable

Description

Determines whether an ANYTYPE parameter was passed a variable or direct constant.

Parameters

variable - An ANYTYPE parameter in a subroutine

Return value

TRUE if the parameter was passed by reference, FALSE if a constant was used.

Remarks

See Also: [TYPEOF](#)

Example usage

```
SUB mysub(v as ANYTYPE)
DEF param as INT
  SELECT TYPEOF(v)
  CASE @TYPEINT
  CASE& @TYPEUINT
  IF ISREF(v) THEN param = ##<INT>v ELSE param = #<INT>v
...
```

12.261 ISSELECTED

Syntax

INT = ISSELECTED(win as WINDOW, id as INT, pos as UINT)

Description

Test for selection in a multi-selection list or combo box.

Parameters

win - Window or dialog containing the control.
id - Identifier of the combo box or list box control
pos - Zero based position for the test.

Return value

Returns 1 if the string at *pos* is currently selected or 0 if it is not selected.

Remarks

Works only on combo and list boxes that have the multi-selection style.

See Also: [GETSELECTED](#), [GETSTRINGCOUNT](#)

Example usage

```
FOR x = 0 TO GETSTRINGCOUNT(win, 7)-1
  IF ISSELECTED(win,7,x) THEN selected[x] = 1 ELSE selected[x] = 0
NEXT x
```

12.262 ISWINDOWCLOSED

Syntax

INT = ISWINDOWCLOSED(win as WINDOW)

Description

Tests a window or dialog to see if it has been closed.

Parameters

win - Window or dialog to test

Return value

Returns TRUE if the window or dialog is currently closed.

Remarks

None.

Example usage

```
WAITUNTIL IsWindowClosed(mainwindow)
```

12.263 LABEL

Syntax

LABEL name

name:

Description

Defines a label in your code for GOTO to jump to.

Parameters

None

Return value

None

Remarks

Labels inside a subroutine are known only to that subroutine. Therefore, multiple subroutines may contain identical labels.

See Also [GOTO](#)

Example usage

```
LABEL mylabel  
...  
mylabel2:
```

12.264 LCASE\$ / WLCASE\$

Syntax

```
STRING = LCASE$(str as STRING)  
WSTRING = WLCASE$(str as WSTRING)
```

Description

Converts a string to all lower case.

Parameters

str - The string to convert.

Return value

A copy of the input string converted to lower case.

Remarks

The input string remains unchanged.

See Also: [UCASE\\$](#)

Example usage

```
PRINT LCASE$("CONVERT TO LOWER CASE PLEASE")
```

12.265 LEAVE

Syntax

```
LEAVE
```

Description

Used to exit a TRY / ENDTRY block of code in the same manner that BREAK and BREAKFOR are used to exit various loops.

Parameters

None

Return value

None

Remarks

See Also: [TRY](#), [ENDTRY](#), [BREAK](#), [BREAKFOR](#), [Exception Handling](#)

Example usage

See [Exception Handling](#)

12.266LEFT\$ / WLEFT\$

Syntax

STRING = LEFT\$(str as STRING,length as INT)

WSTRING = WLEFT\$(str as WSTRING,length as INT)

Description

Extracts the first (leftmost) characters of a string and returns a copy of the extracted string.

Parameters

str - The string to extract characters from.

length - Leftmost count of characters to extract.

Return value

A copy of the extracted substring.

Remarks

Length can be zero in which case an empty string is returned.

See Also: [RIGHT\\$](#), [MID\\$](#)

Example usage

```
a$ = "The Brown Cow"  
IF LEFT$(a$, 3) = "The" THEN PRINT "I understand!"
```

12.267LEN

Syntax

size = LEN(variable)

size = LEN(UDT)

Description

Returns the length in bytes of a variable or UDT.

Parameters

variable - Any defined variable.

UDT - The name of a user data type as specified in the TYPE statement

Return value

Depends on the type of variable:

FILE, BFILE - Returns the length of the file, file must have been opened with OPENFILE.

MEMORY - Returns the size of the memory allocated with ALLOCMEM.

UDT - Returns the actual size a UDT takes in memory using the packing value specified in the TYPE statement

STRING - Returns the string length, not the defined length.

All others - Returns the total size in bytes. For arrays this returns the total size of the array.

Remarks

For a MEMORY variable only memory obtained with ALLOCMEM will return a size.

The LEN statement can be used in expressions requiring a constant value, known at compile time.

Almost any expression passed to LEN is supported, but string:

```
const MYCONST = len(1) + len(int) + len(myarray[1,2,3,4,5]) + len(my_int_variable) + 1
```

A function name passed to LEN will return the size of this subroutine. But the subroutine must exist in current source file (or in any included file). In this case, LEN will complete while assembling.

See Also: [ALLOCMEM](#), [DEF / DIM](#), [TYPE](#), [SIZEOF](#)

Example usage

```
mystring = "This is a test"
PRINT LEN(mystring)
```

12.268LINE**Syntax**

LINE(win as WINDOW,x1 as UINT,y1 as UINT,x2 as UINT,y2 as UINT,OPT crColor as UINT)

Description

Draws a straight line in the window.

Parameters

win - Window to draw the line into.

x1, y1 - Starting coordinates of the line.

x2, y2 - Ending coordinates of the line.
crColor - Optional line color.

Return value

None

Remarks

If *crColor* is not supplied then the line is drawn in the foreground color set by FRONTPEN.

See Also: [LINETO](#) , [FRONTPEN](#)

Example usage

```
LINE mywnd, 0, 0, 50, 50, RGB(0,255,0)
```

12.269LINETO

Syntax

LINETO(win as WINDOW,x1 as UINT,y1 as UINT,OPT crColor as UINT)

Description

Draws a line from the current pen position to the position specified.

Parameters

win - Window to draw line into.
x1, y1 - Ending coordinates for line.
crColor - Optional line color.

Return value

None.

Remarks

The current pen position is updated directly by the MOVE command. If *crColor* is not supplied then the line is drawn in the foreground color set by FRONTPEN.

See Also: [MOVE](#), [LINE](#), [FRONTPEN](#)

Example usage

```
MOVE wnd, 10,10  
LINETO wnd, 50,50  
LINETO wnd, 50,100
```

12.270ListAdd

Syntax

POINTER = ListAdd(list as POINTER,pData as POINTER)

Description

Adds a new node to a linked list

Parameters

list - A list created by the ListCreate function.

pData- Any pointer to data to use as the data member of the new node.

Return value

For convenience this function returns the *pData* pointer.

Remarks

Node is added to the tail of the list. Use ListAddHead to add a new node to the head of the list.

See Also: [ListCreate](#), [ListAddHead](#)

Example usage

```
mylist = ListCreate()  
FOR x = 0 to 10  
    temp = ListAdd(mylist, NEW(INT,1))  
    #<INT>temp = x  
NEXT x
```

12.271ListAddHead

Syntax

POINTER = ListAddHead(list as POINTER,pData as POINTER)

Description

Adds a new node to a linked list. Node is added to the head of the list.

Parameters

list - A list created by the ListCreate function.

pData- Any pointer to data to use as the data member of the new node.

Return value

For convenience this function returns the *pData* pointer.

Remarks

See Also: [ListCreate](#), [ListAdd](#)

Example usage

```
mylist = ListCreate()  
FOR x = 0 to 10  
    temp = ListAddHead(mylist, NEW(INT,1))  
    #<INT>temp = x  
NEXT x
```

12.272ListCreate

Syntax

```
POINTER = ListCreate
```

Description

Creates a new linked list.

Parameters

None

Return value

A pointer to the head of an empty linked list.

Remarks

Add nodes to the list using `ListAdd` or `ListAddHead`.

See Also: [ListAdd](#), [ListAddHead](#)

Example usage

```
mylist = ListCreate()
FOR x = 0 to 10
    temp = ListAdd(mylist, NEW(INT,1))
    #<INT>temp = x
NEXT x
```

12.273 ListGetData

Syntax

```
POINTER = ListGetData(pos as POINTER)
```

Description

Retrieves the data pointer from a node in a linked list

Parameters

pos - A node pointer returned by `ListGetFirst` or `ListGetNext`

Return value

A pointer to the data added with `ListAdd` or `ListAddHead`

Remarks

Its important to remember the relationship between a node and a data pointer. The node is create by adding data to the list, the data is a pointer supplied by you to some arbitrary information you wish to store in the linked list. `ListGetFirst` and `ListGetNext` return the pointer to the node and `ListGetData` returns a pointer to your data stored in that node.

See Also: [ListGetFirst](#), [ListGetNext](#), [ListAdd](#), [ListAddHead](#)

Example usage

```
pos = ListGetFirst(mylist)
WHILE pos <> 0
```

```
data = ListGetData(pos)
PRINT #<INT>data,
pos = ListGetNext(pos)
ENDWHILE
```

12.274 ListGetFirst

Syntax

POINTER = ListGetFirst(list as POINTER)

Description

Retrieves the head node in a linked list.

Parameters

list - A list created by the ListCreate function.

Return value

The first node of the linked list or NULL if no nodes exist.

Remarks

The returned node can subsequently be used in the ListGetData and ListGetNext functions.

See Also: [ListGetData](#), [ListGetNext](#), [ListCreate](#)

Example usage

```
pos = ListGetFirst(mylist)
WHILE pos <> 0
    data = ListGetData(pos)
    PRINT #<INT>data,
    pos = ListGetNext(pos)
ENDWHILE
```

12.275 ListGetNext

Syntax

POINTER = ListGetNext(pos as POINTER)

Description

Retrieves the next node in sequence.

Parameters

pos - The previous node returned by ListGetNext or ListGetFirst.

Return value

A pointer to the next node.

Remarks

See Also: [ListGetFirst](#)

Example usage

```
pos = ListGetFirst(mylist)
WHILE pos <> 0
    data = ListGetData(pos)
    PRINT #<INT>data,
    pos = ListGetNext(pos)
ENDWHILE
```

12.276ListRemove

Syntax

POINTER = ListRemove(pos as POINTER,OPT bDelete=0 as INT)

Description

Removes a node from a list.

Parameters

pos - Node to remove as returned by ListGetNext or ListGetFirst.

bDelete - Optional. If 1 then the DELETE function is called for the data pointer.

Return value

The next node in the list or NULL if this was the tail node.

Remarks

See Also: [ListGetNext](#), [ListGetFirst](#), [ListRemoveAll](#)

Example usage

```
'remove a data item
pos = ListGetFirst(mylist)
WHILE pos <> 0
    data = ListGetData(pos)
    IF #<INT>data = 5
        pos = ListRemove(pos, TRUE)
    ELSE
        pos = ListGetNext(pos)
    ENDF
ENDWHILE
```

12.277ListRemoveAll

Syntax

ListRemoveAll(list as POINTER,OPT bDelete=0 as INT)

Description

Removes all nodes from the list and deletes the list.

Parameters

list - The linked list returned by ListCreate

bDelete - Optional. If 1 then DELETE is called for all data pointers in the list.

Return value

None

Remarks

The list pointer is invalid when this function completes and cannot be used again unless ListCreate is called to create a new list.

See Also: [ListRemove](#), [ListCreate](#)

Example usage

```
ListRemoveAll(mylist, TRUE)
```

12.278LOADIMAGE

Syntax

UINT = LOADIMAGE(identifier, type as INT)

Description

Loads an image from a file or resources and returns a handle for that image.

Parameters

identifier - Filename, resource name or resource ID to image.

type - type of image, see remarks.

Return value

Handle to the loaded image. Handle can be subsequently be used in the SHOWIMAGE, SETCURSOR or SETICON functions.

Remarks

Type is a numeric value defining what kind of image to load.

The valid values for type are:

@IMGBITMAP - bitmap (*.bmp)

@IMGICON - Icon (*.ico)

@IMGCURSOR - Cursor (*.cur)

@IMGEMF - Enhanced meta file (*.emf)

@IMGSCALABLE - scalable bitmap, JPEG (*.jpg) or GIF (*.gif) files.

If a filename is specified as the *identifier* then the image is loaded from disk. IW BASIC can also load bitmaps, icons and cursors directly from the executables resources. Enhanced meta files

cannot be loaded from the resource table. Scalable images may be loaded from resources as custom type of RTIMAGE. The resource *identifier* is either the string or integer identifier of a resource compiled with the project.

When done with an image use the DELETEIMAGE statement on the returned handle.

See Also: [SHOWIMAGE](#), [SETCURSOR](#), [SETICON](#), [DELETEIMAGE](#)

Example usage

```
mybmp = LOADIMAGE(GETSTARTPATH + "picture.bmp", @IMGBITMAP)
```

12.279LOADMENU

Syntax

```
INT = LOADMENU(win as WINDOW, identifier)
```

Description

Loads a menu from resources and sets the menu bar for the dialog or window.

Parameters

win - Window to set the newly loaded menu.

identifier - Resource name or numeric identifier.

Return value

The return value is 0 if the menu could not be loaded from the resources.

Remarks

See Also: [LOADRESOURCE](#)

Example usage

```
LOADMENU(mywnd, "mymenu")
```

Example resource:

```
BEGIN
  POPUP "&File"
  BEGIN
    MENUITEM "&New\tCtrl+N", 25
    MENUITEM "&Open...\tCtrl+O", 26
    MENUITEM SEPARATOR
    MENUITEM "P&rint Setup...", 27
    MENUITEM SEPARATOR
    MENUITEM "Recent File", 28
    MENUITEM SEPARATOR
    MENUITEM "E&xit", 29
  END
  POPUP "&Help"
  BEGIN
    MENUITEM "&About ChartCraft...", 30
  END
```

END

12.280LOADRESOURCE

Syntax

INT = LOADRESOURCE(identifier, type, var)

Description

Loads a resource from the executable and places either a copy of the resource, or an integer pointer to the resource, in the variable specified.

Parameters

identifier - Resource name or numeric identifier.

type - Type name or numeric identifier.

var - STRING, INT, UINT, POINTER or MEMORY variable to store the resource contents.

Return value

Returns 0 on failure, 1 on success.

Remarks

If *var* is a *STRING* (or *ISTRING*) variable the contents of the resource will be copied into the string. Useful for embedded text resources.

If *var* is an *INT* / *UINT* variable then a handle to the locked resource is returned. Useful for API calls.

If *var* is a *MEMORY* variable then memory is allocated for the resource and resource copied into the memory. You can then use *READMEM* to read the resource data. *LEN(variable)* will return the length of the resource in this case. You must free the memory returned with the *FREEMEM* statement. Using a *MEMORY* variable that has been allocated with *ALLOCMEM* will result in memory leaks in your program.

If *var* is a *POINTER* then a pointer to the locked resource is returned. You can read the resource using standard type casting.

The standard windows resource types can be loaded by specifying one of the following for the *type* parameter:

@RESCURSOR

@RESBITMAP

@RESICON

@RESMENU

@RESDIALOG

@RESSTRING

@RESACCEL

@RESDATA
@RESMESSAGETABLE
@RESGROUPCURSOR
@RESGROUPICON
@RESVERSION

See Also: [LOADIMAGE](#), [LOADMENU](#)

Example usage

```
IF LOADRESOURCE("mydata",@RESDATA,pData)
    PRINT #<UINT>pData
ENDIF
```

12.281LOADTOOLBAR

Syntax

INT = LOADTOOLBAR(win as WINDOW,handle as ANYTYPE,id as UINT,tbarray as POINTER,array_size as INT,style as INT)

Description

Loads and creates a tool bar control. Bitmap for the toolbar can be loaded from a file or directly from resources.

Parameters

win - Window or dialog to show the toolbar.
handle - Filename, resource name or resource numeric identifier.
id - Control identifier for the toolbar.
tbarray - Integer array of button identifiers.
array_size - Number of elements of the array.
style - Toolbar style.

Return value

Returns 0 on failure, 1 on success

Remarks

See Also: [CONTROLCMD](#), Using Toolbars

Example usage

See *loadtoolbar.iwb*, *verticaltoolbar.iwb*

12.282LOCATE

Syntax

LOCATE(line as INT, column as INT)

Description

Positions the caret in the console window.

Parameters

line - The line in the console window to place the caret.

column - The character position to place the caret.

Return value

None

Remarks

OPENCONSOLE must have been executed before this statement.

Example usage

```
OPENCONSOLE
LOCATE 10,1
PRINT "Position 10,1"
LOCATE 10,50
PRINT "Position 10,50"
LOCATE 1,1
PRINT "Position 1,1"
LOCATE 1,50
PRINT "Position 1,50"
LOCATE 12,1
PRINT "Press Any Key To Close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

12.283 LOG10

Syntax

DOUBLE = LOG10(num as DOUBLE)

Description

Calculate the base 10 logarithm of a number if successful

Parameters

num - Value whose logarithm is to be found

Return value

The logarithm of a number unless the number is less than or equal to zero. If the input is less than zero then the result is indefinite. If the input is equal to zero then the result is infinite and cannot be represented.

Remarks

See also [LOG](#)

Example usage

```
PRINT LOG10(9000.0)
```

12.284LOG

Syntax

DOUBLE = LOG(num as DOUBLE)

Description

Calculates the natural logarithm of a number if successful

Parameters

num - Value whose logarithm is to be found

Return value

The logarithm of a number unless the number is less than or equal to zero. If the input is less than zero then the result is indefinite. If the input is equal to zero then the result is infinite and cannot be represented.

Remarks

See also [LOG10](#)

Example usage

```
PRINT LOG(9000.0)
```

12.285LTRIM\$ / WLTRIM\$

Syntax

STRING = LTRIM\$(str as STRING)
WSTRING = WLTRIM\$(str as WSTRING)

Description

Trims leading whitespace characters from the string.

Parameters

str - The input string

Return value

A copy of the input string minus any leading whitespace characters.

Remarks

Whitespace as defined by this function includes space and tab characters.

See Also: [RTRIM\\$](#)

Example usage

```
PRINT LTRIM$ ("      Remove leading spaces")
```

12.286MEMORY

Syntax

memory = MEMORY(num)

Description

Converts a value to a MEMORY pointer.

Parameters

num - A numeric expression

Return value

A memory pointer value.

Remarks

If num is not a valid memory pointer the results will be unpredictable.

Example usage

```
MEMORY(ptr)
```

12.287MENUITEM

Syntax

MENUITEM text as STRING, style as UINT, id as INT

Description

Defines a menu item in a menu definition macro.

Parameters

text - Text of the menu item.

style - Style flags of menu item.

id - Identifier of menu item.

Return value

None

Remarks

A menu item is the selectable portion of a menu. The only style flags currently built in are @MENCHECK and @MENUDISABLE for an initially checked menu item or initially disabled menu item. *id* must be a unique identifier greater than 0.

See Also: [BEGINMENU](#), [ENDMENU](#), [MENUTITLE](#), [BEGINPOPUP](#), [ENDPOPUP](#), [SEPARATOR](#), [BEGININSERTMENU](#)

Example usage

```
BEGINMENU win
```

```
MENUTITLE "&File"  
MENUITEM "Open",0,1  
MENUITEM "Close",0,2  
BEGINPOPOP "Save As..."  
    MENUITEM "Ascii",0,3  
    MENUITEM "Binary",0,4  
ENDPOPOP  
SEPARATOR  
MENUITEM "&QUIT",0,5  
MENUTITLE "&Edit"  
MENUITEM "Cut",0,6  
ENDMENU
```

12.288MENUTITLE

Syntax

MENUTITLE title as STRING

Description

Defines a new menu title (pull down) in a menu definition macro

Parameters

title - The menu title

Return value

None

Remarks

Creates the title on the menu bar for the pull down. All MENUITEM statements after this will appear in this menu.

See Also: [BEGINMENU](#), [ENDMENU](#), [MENUITEM](#), [BEGINPOPOP](#), [ENDPOPOP](#), [SEPARATOR](#), [BEGININSERTMENU](#)

Example usage

```
BEGINMENU win  
    MENUTITLE "&File"  
    MENUITEM "Open",0,1  
    MENUITEM "Close",0,2  
    BEGINPOPOP "Save As..."  
        MENUITEM "Ascii",0,3  
        MENUITEM "Binary",0,4  
    ENDPOPOP  
    SEPARATOR  
    MENUITEM "&QUIT",0,5  
    MENUTITLE "&Edit"  
    MENUITEM "Cut",0,6  
ENDMENU
```

12.289MESSAGEBOX

Syntax

INT = MESSAGEBOX(win as POINTER,text as STRING,caption as STRING,OPT flags as UINT)

Description

Displays a standard message box.

Parameters

win - Window or dialog to use as the parent, can be NULL.

text - Text to display in the body.

caption - Title bar caption text.

flags - Optional. Style of the message box.

Return value

Return based on style of the message box.

Remarks

Flags are a combination of a message box type and an icon identifier or'ed together. Default is @MB_OK.

Message box types:

@MB_OK - One button "OK", no return value.

@MB_OKCANCEL - Two buttons, "OK" and "CANCEL".

@MB_ABORTRETRYIGNORE - Three buttons "ABORT", "RETRY", and "IGNORE"

@MB_YESNOCANCEL - Three buttons "YES", "NO", and "CANCEL"

@MB_YESNO - Two buttons "YES" and "NO"

@MB_RETRYCANCEL - Two buttons "RETRY" and "CANCEL"

@MB_CANCELTRYCONTINUE- Three buttons "CANCEL", "TRY AGAIN", and "CONTINUE"

Icon identifiers:

@MB_ICONEXCLAMATION

@MB_ICONINFORMATION

@MB_ICONQUESTION

@MB_ICONSTOP

Return values:

@IDOK

@IDCANCEL

@IDABORT

@IDRETRY

@IDIGNORE

@IDYES

@IDNO

@IDTRYAGAIN

@IDCONTINUE

Example usage

```
IF MESSAGEBOX(mywnd,"Can't read file","error",@MB_ICONSTOP | @MB_RETRYCANCEL) = @IDRE?  
...  
ENDIF
```

12.290MID\$ / WMID\$

Syntax

STRING = MID\$(str as STRING,start as INT,OPT count=-1 as INT)
WSTRING = WMID\$(str as WSTRING,start as INT,OPT count=-1 as INT)

Description

Extracts zero or more characters from any position in the specified input string.

Parameters

str - The input string to extract characters from/

start - Ones based starting position of the extraction.

count - Optional. Number of characters to extract. If omitted then all characters from the starting position to the end of the string are included in the extracted substring.

Return value

A copy of the extracted substring.

Remarks

See Also: [LEFTS](#), [RIGHTS](#)

Example usage

```
A$ = "This is a test"  
IF MID$(A$, 6, 2) = "is" THEN PRINT "cool"  
PRINT MID$(A$, 9)
```

12.291MILLISECS

Syntax

MILLISECS()

Description

Returns the number of milliseconds elapsed since Windows was booted.

Parameters

None

Return value

None

Remarks

Convenience alias to the API function `GetTickCount`. Note that this must be used like a function with empty parenthesis.

Example usage

```
PRINT MILLISECS ( )
```

12.292MODIFYEXSTYLE

Syntax

`MODIFYEXSTYLE(win as WINDOW,add as UINT,remove as UINT,OPT id=0 as INT)`

Description

Adds or removes extended styles from a window, dialog or control.

Parameters

win - Window or dialog.

add - Extended styles to add.

remove - Extended styles to remove.

id - Optional control ID.

Return value

None

Remarks

If *id* is specified then the extended style of a control is changed whose parent is the window or dialog specified. You can add styles, remove styles, or both at the same time. Use 0 for an unused style parameter.

See Also: [MODIFYSTYLE](#), [REDRAWFRAME](#)

Example usage

```
' Remove the 3D border from an edit control
CONST WS_EX_CLIENTEDGE = 0x200
MODIFYEXSTYLE mywin, 0, WS_EX_CLIENTEDGE, 2
REDRAWFRAME mywin, 2
```

12.293MODIFYSTYLE

Syntax

`MODIFYSTYLE(win as WINDOW,add as UINT,remove as UINT,OPT id=0 as INT)`

Description

Adds or removes styles from a window, dialog or control.

Parameters

win - Window or dialog.

add - Styles to add.
remove - Styles to remove.
id - Optional control ID.

Return value

None

Remarks

If *id* is specified then the style of a control is changed whose parent is the window or dialog specified. You can add styles, remove styles, or both at the same time. Use 0 for an unused style parameter.

See Also: [MODIFYEXSTYLE](#), [REDRAWFRAME](#)

Example usage

```
' Adds a minimize box to the window, removes the maximize box
MODIFYSTYLE mywin, @MINBOX, @MAXBOX
' Show the changes
REDRAWFRAME mywin
```

12.294MOVE

Syntax

MOVE(win as WINDOW,x as INT, y as INT)

Description

Updates the pen position in the window.

Parameters

win - Window.
x, *y* - New coordinates, in pixels, for the pen position.

Return value

None

Remarks

The pen position is used by drawing commands and PRINT to determine where to draw in the window.

Example usage

```
MOVE 10,20
PRINT mywnd, "Pixel location 10,20"
```

12.295NEW

Syntax

POINTER = NEW(type, size as INT)

POINTER = NEW type

POINTER = NEW type[size as INT]

Description

Creates a dynamic variable in memory.

Parameters

type - Any built in type such as INT, STRING, etc. or a UDT name.

size - If greater than one will create a single dimensional array of the specified type.

Return value

A pointer to the newly created variable

Remarks

The pointer returned can not be an autodefined one. You must define the pointer first.

See Also: [DELETE](#), Using Pointers

Example usage

```
DEF pInt as POINTER
pInt = NEW(INT,10)
#<INT>pInt[0] = 7
#<INT>pInt[5] = 8
DELETE pInt
```

12.296NEXT

Syntax

NEXT OPT var as ANYTYPE

Description

Terminates a FOR/NEXT or FOR/EACH loop

Parameters

var - Must match the variable specified in the FOR statement. Not used for a FOR/EACH loop

Return value

None

Remarks

See Also: [FOR](#)

Example usage

```
FOR x = 1 to 10
PRINT x
NEXT x
```

```
FOR mydata = EACH mylist AS STRING
PRINT #mydata
NEXT
```

12.297NOT

Syntax

return = NOT(num)

Description

Returns the ones compliment of the number.

Parameters

num - CHAR, WORD, INT, UINT, INT64, UINT64 parameter

Return value

The ones compliment otherwise known as a bitwise NOT.

Remarks

In binary NOT turns all 0's into 1's and all 1's into 0's

Example usage

```
PRINT NOT (1)
```

12.298ONCONTROL

Syntax

ONCONTROL(window win,int id,int notify,uint callback)

Description

Installs a handler subroutine to process a message.

Parameters

win - Window or dialog containing the control.

id - Controls identifier.

notify - The notification code to process.

callback - Address of message handler subroutine. Use the & operator to specify.

Return value

None

Remarks

The callback subroutine has the template of:

```
SUB name( ),INT
```

When the window or dialog receives a message it first checks to see if you have installed a handler for that message, and will call that subroutine before the main message handler, if any. The handler can only be installed after the window has been created.

Your subroutine should return FALSE (0) to indicate that other message handlers can be called.

The message handler installers are an alternative way of processing messages in your program, and can be used to better organize your code. Separate message handler subroutines are also used by visual designers.

Example usage

```

window win
openwindow win,0,0,640,400,@SIZE|@HIDDEN,NULL,"Handlers test",NULL
control win,@edit,"",320-50,200-12,100,24,@cteditautoh,99
BeginMenu win
    MenuTitle "Action"
        MenuItem "Quit",0,100
EndMenu
CenterWindow win
ShowWindow win,@SWSHOW

OnMessage win,@IDCLOSEWINDOW,&DoEndProgram
OnMessage win,@IDSIZE,&DoWindowSized
OnMenuPick win,100,&DoEndProgram
OnControl win,99,@ENCHANGE,&DoEditChanged
OnControl win,99,@ENSETFOCUS,&DoEditFocus
OnControl win,99,@ENKILLFOCUS,&DoEditFocus
SetFocus win,99

WaitUntil IsWindowClosed(win)
End

SUB DoEndProgram(),INT
    CloseWindow win
    Return 0
ENDSUB

SUB DoWindowSized(),INT
    int l,t,w,h
    GetSize win,l,t,w,h
    Move win,10,10
    print win,using("0##### 0#####",w,h)
    Return 0
ENDSUB

SUB DoEditChanged(),INT
    move win,320-50,200+14
    print win,GetControlText(win,99)+space$(30)
    Return 0
ENDSUB

SUB DoEditFocus(),INT
    if @notifycode = @ensetfocus
        SetControlColor win,99,0,rgb(255,255,0)
    else if @notifycode = @enkillfocus

```

```
        SetControlColor win,99,0,rgb(255,255,255)
    endif
    Return 0
ENDSUB
```

12.299ONMENU PICK

Syntax

ONMENU PICK(window win,int id,uint callback)

Description

Installs a handler subroutine to process a menu message.

Parameters

win - Window or dialog containing the control.

id - Menu identifier.

callback - Address of message handler subroutine. Use the & operator to specify.

Return value

None

Remarks

The callback subroutine has the template of:

```
SUB name( ),INT
```

When the window or dialog receives a message it first checks to see if you have installed a handler for that message, and will call that subroutine before the main message handler, if any. The handler can only be installed after the window has been created.

Your subroutine should return FALSE (0) to indicate that other message handlers can be called.

The message handler installers are an alternative way of processing messages in your program, and can be used to better organize your code. Separate message handler subroutines are also used by visual designers.

Example usage

```
window win
openwindow win,0,0,640,400,@SIZE|@HIDDEN,NULL,"Handlers test",NULL
control win,@edit,"",320-50,200-12,100,24,@cteditautoh,99
BeginMenu win
    MenuTitle "Action"
        MenuItem "Quit",0,100
EndMenu
CenterWindow win
ShowWindow win,@SWSHOW

OnMessage win,@IDCLOSEWINDOW,&DoEndProgram
```

```

OnMessage win,@IDSIZE,&DoWindowSized
OnMenuPick win,100,&DoEndProgram
OnControl win,99,@ENCHANGE,&DoEditChanged
OnControl win,99,@ENSETFOCUS,&DoEditFocus
OnControl win,99,@ENKILLFOCUS,&DoEditFocus
SetFocus win,99

WaitUntil IsWindowClosed(win)
End

SUB DoEndProgram(),INT
    CloseWindow win
    Return 0
ENDSUB

SUB DoWindowSized(),INT
    int l,t,w,h
    GetSize win,l,t,w,h
    Move win,10,10
    print win,using("0##### 0#####",w,h)
    Return 0
ENDSUB

SUB DoEditChanged(),INT
    move win,320-50,200+14
    print win,GetControlText(win,99)+space$(30)
    Return 0
ENDSUB

SUB DoEditFocus(),INT
    if @notifycode = @ensetfocus
        SetControlColor win,99,0,rgb(255,255,0)
    else if @notifycode = @enkillfocus
        SetControlColor win,99,0,rgb(255,255,255)
    endif
    Return 0
ENDSUB

```

12.300ONMESSAGE

Syntax

ONMESSAGE(window win,int message,uint callback)

Description

Installs a handler subroutine to process a window or dialog message.

Parameters

win - Window or dialog containing the control.

message - Message identifier.

callback - Address of message handler subroutine. Use the & operator to specify.

Return value

None

Remarks

The callback subroutine has the template of:

```
SUB name( ),INT
```

When the window or dialog receives a message it first checks to see if you have installed a handler for that message, and will call that subroutine before the main message handler, if any. The handler can only be installed after the window has been created, or dialog shown.

Your subroutine should return FALSE (0) to indicate that other message handlers can be called.

The message handler installers are an alternative way of processing messages in your program, and can be used to better organize your code. Separate message handler subroutines are also used by visual designers.

The message @IDCREATE cannot be processed with a message handler like this, it has to be processed in the main handler subroutine.

Example usage

```

window win
openwindow win,0,0,640,400,@SIZE|@HIDDEN,NULL,"Handlers test",NULL
control win,@edit,"",320-50,200-12,100,24,@cteditautoh,99
BeginMenu win
    MenuTitle "Action"
        MenuItem "Quit",0,100
EndMenu
CenterWindow win
ShowWindow win,@SWSHOW

OnMessage win,@IDCLOSEWINDOW,&DoEndProgram
OnMessage win,@IDSIZE,&DoWindowSized
OnMenuPick win,100,&DoEndProgram
OnControl win,99,@ENCHANGE,&DoEditChanged
OnControl win,99,@ENSETFOCUS,&DoEditFocus
OnControl win,99,@ENKILLFOCUS,&DoEditFocus
SetFocus win,99

WaitUntil IsWindowClosed(win)
End

SUB DoEndProgram(),INT
    CloseWindow win
    Return 0
ENDSUB

SUB DoWindowSized(),INT
    int l,t,w,h
    GetSize win,l,t,w,h
    Move win,10,10
    print win,using("0##### 0#####",w,h)
    Return 0
ENDSUB

```

```

SUB DoEditChanged(),INT
    move win,320-50,200+14
    print win,GetControlText(win,99)+space$(30)
    Return 0
ENDSUB

SUB DoEditFocus(),INT
    if @notifycode = @ensetfocus
        SetControlColor win,99,0,rgb(255,255,0)
    else if @notifycode = @enkillfocus
        SetControlColor win,99,0,rgb(255,255,255)
    endif
    Return 0
ENDSUB

```

12.301ONEXIT

Syntax

ONEXIT(*fn* as UINT,*pData* as POINTER), INT

Description

Adds a subroutine to the on-exit list.

Parameters

fn - Address of subroutine.

pData - Pointer to user data or NULL

Return value

INT Note: This value is not used but must be is the subroutine return.

Remarks

The functions in the on-exit list are called in LIFO (Last In First Out) order when your program exits. The DECLARE for an on-exit function must include one parameter of type POINTER (May be NULL or 0). A subroutine can be added more than once.

Example usage

```

OPENCONSOLE

ONEXIT(&myfunc1,NULL)
ONEXIT(&myfunc2,0)
ONEXIT(&myfunc1,"hello")

waitcon
END

SUB myfunc1(pData as POINTER),int
    IF pData = NULL
        MESSAGEBOX 0,"myfunc1","TEST1"
    ELSE
        MESSAGEBOX 0,"myfunc1",#<STRING>pData
    ENDIF
    RETURN 0

```

```
ENDSUB  
  
SUB myfunc2(pData as POINTER),int  
    MESSAGEBOX 0,"myfunc2","TEST2"  
    RETURN 0  
ENDSUB
```

12.302OPENCONSOLE

Syntax

```
OPENCONSOLE
```

Description

Opens the text-only console window for output.

Parameters

None

Return value

None

Remarks

The system only allows one text console per process. If a console window is already opened for output then this command will do nothing.

See Also: [CLOSECONSOLE](#)

Example usage

```
OPENCONSOLE  
PRINT "hello"  
DO:UNTIL INKEY$ <> ""  
CLOSECONSOLE  
END
```

12.303OPENFILE

Syntax

```
INT = OPENFILE(fptr as FILE | BFILE, filename as STRING, mode as STRING)
```

Description

Opens a file or stream for reading, writing or appending.

Parameters

fptr - A FILE or BFILE variable.

filename - Name of file to open, or name of a shared resource such as "LPT1:".

mode - "W" for writing, "A" for appending, "R" for reading, "R+" for read/write access on existing

files.

Return value

Returns 0 on success, 1 on failure.

Remarks

Use a BFILE variable to open the file in binary mode. Use LEN on an open file variable to determine the size of the file.

See Also: [CLOSEFILE](#), [READ](#), [WRITE](#), [GET](#), [PUT](#)

Example usage

```
DEF myfile as BFILE
IF OPENFILE(myfile, "C:\\temp\\temp.txt", "w") = 0
    WRITE myfile, 0xFFFF
    CLOSEFILE myfile
ENDIF
```

12.304 OPENPRINTER

Syntax

UINT = OPENPRINTER(name as STRING,title as STRING,mode as STRING)

Description

Opens a printer for writing.

Parameters

name - Name of printer returned by GETDEFAULTPRINTER or PRTDIALOG.

title - Title of print job to display in the print manager.

mode - "RAW" or "TEXT"

Return value

A handle to the open printer or 0 if the printer could not be found.

Remarks

Not all printers support direct text printing. Printers sold as "GDI Only" do not include built in fonts and will not work with the WRITEPRINTER command. For "GDI Only" printers use the PRINTWINDOW command instead.

See Also: [WRITEPRINTER](#), [CLOSEPRINTER](#), [PRTDIALOG](#), [GETDEFAULTPRINTER](#)

Example usage

```
prtname = GETDEFAULTPRINTER
hPrinter = OPENPRINTER(prtname, "Job 1", "TEXT")
IF hPrinter
    WRITEPRINTER hPrinter, "This is line 1\n"
    CLOSEPRINTER hPrinter
ENDIF
```

12.305OPENWINDOW

Syntax

INT = OPENWINDOW(win as WINDOW,l as INT,t as INT,w as INT,h as INT,flags as UINT,parent as POINTER,title as STRING,procedure as UINT)

Description

Creates a new window.

Parameters

win - WINDOW variable.

l, t, w, h - Position and dimensions of new window.

flags - Style flags of window.

parent - Windows parent or NULL.

title - Caption text for window.

procedure - Address of subroutine to process messages for this window.

Return value

Returns 0 on failure, 1 if window was successfully created.

Remarks

See Also: [CLOSEWINDOW](#), Creating Windows

Example usage

```
REM define a window variable
DEF w1 as WINDOW
REM open the window
OPENWINDOW w1,0,0,350,350,@MINBOX|@MAXBOX|@SIZE,0,"Simple Window",&main
REM print a message
PRINT w1,"Hello World"
REM when w1 = 0 the window has been closed
WAITUNTIL w1 = 0
END
'----
REM every time there is a message for our window
REM the operating system will GOSUB here
SUB main( ),INT
  IF @CLASS = @IDCLOSEWINDOW
    REM closes the window and sets w1 = 0
    CLOSEWINDOW w1
  ENDIF
RETURN 0
ENDSUB
```

12.306PagerControl

Syntax

UINT = PagerControl(win as WINDOW,l as INT,t as INT,w as INT,h as INT,flags as INT,exStyle as INT,id as UINT)

Description

Creates a pager control. A *pager control* is a window container that is used with a window that does not have enough display area to show all of its content.

Parameters

win - Parent window or dialog containing the control.

l, t, w, h - Position and dimension of the control.

flags - Styles for the control.

exStyle - Extended window styles for the control

id - Control identifier.

Return value

If used with a window the return value is the handle of the control. If used with a dialog the return value is 0 and the handle to the control can be obtained with GETCONTROLHANDLE during the @IDINITDIALOG message

Remarks

The control can also be created using CONTROLEX with a window class of "SysPager".

Styles

@PGS_VERT - Creates a pager control that can be scrolled vertically. This is the default.

@PGS_HORZ - Creates a pager control that can be scrolled horizontally. This style and the

@PGS_VERT style are mutually exclusive and cannot be combined.

@PGS_AUTOSCROLL - The pager control will scroll when the user hovers the mouse over one of the scroll buttons.

@PGS_DRAGNDROP - The contained window can be a drag-and-drop target. The pager control will automatically scroll if an item is dragged from outside the pager over one of the scroll buttons.

Example usage

```
PagerControl cp,0,0,w,20,@PGS_HORZ,0,IDPAGER
pcSetChild cp,IDPAGER,IDHEADER
```

12.307pcSetPos

Syntax

```
pcSetPos(win as WINDOW,id as UINT,pos as INT)
```

Description

Sets the current scroll position of the pager control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

pos - Value that specifies the new scroll position, in pixels.

Return value

None.

Remarks

None.

Example usage

```
pcSetPos demo, 600, 50
```

12.308 pcSetChildHwnd

Syntax

```
pcSetChildHwnd(win as WINDOW, id as UINT, hwndChild as UINT)
```

Description

Sets the contained window for the pager control. This message will not change the parent of the contained window; it only assigns a window handle to the pager control for scrolling.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

hwndChild - Handle to the window to be contained.

Return value

None

Remarks

For Emergence created controls it is easier to use the [pcSetChild](#) command.

Example usage

```
hChild = GetControlHandle(demo, childID)
pcSetChildHwnd(demo, 500, hChild)
```

12.309 pcSetChild

Syntax

```
pcSetChild(win as WINDOW, id as UINT, child as UINT)
```

Description

Sets the contained control for the pager control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

child - ID of the control to be contained.

Return value

None

Remarks

None

Example usage

```
PagerControl cp,0,0,w,20,@PGS_HORZ,0,IDPAGER  
pcSetChild cp,IDPAGER,IDHEADER
```

12.310 pcSetButtonSize

Syntax

pcSetButtonSize(win as WINDOW,id as UINT,size as INT)

Description

Sets the current button size for the pager control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

size - Size of the pager buttons in pixels.

Return value

None.

Remarks

If the pager control has the @PGS_HORZ style, the button size determines the width of the pager buttons. If the pager control has the @PGS_VERT style, the button size determines the height of the pager buttons. By default, the pager control sets its button size to three-fourths of the width of the scroll bar.

There is a minimum size to the pager button, currently 12 pixels. However, this can change so you should not depend on this value.

Example usage

```
pcSetButtonSize cp,IDPAGER,20
```

12.311 pcSetBorderSize

Syntax

pcSetBorderSize(win as WINDOW,id as UINT,size as INT)

Description

Sets the current border size for the pager control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

size - Size of the border, in pixels.

Return value

None

Remarks

The border should not be larger than the pager button or less than zero. If *size* is too large, the border will be drawn the same size as the button. If *size* is negative, the border size will be set to zero.

Example usage

```
pcSetBorderSize cp, IDPAGER, 1
```

12.312pcSetBackColor

Syntax

```
pcSetBackColor(win as WINDOW, id as UINT, clr as UINT)
```

Description

Sets the current background color for the pager control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

clr - RGB color value.

Return value

None

Remarks

By default, the pager control will use the system button face color as the background color. This is the same color that can be retrieved by calling `GetSysColorBrush` with `COLOR_BTNFACE`

Example usage

```
pcSetBackColor demo, 500, RGB(127,127,127)
```

12.313pcRecalcSize

Syntax

pcRecalcSize(win as WINDOW,id as UINT)

Description

Forces the pager control to recalculate the size of the contained window.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

None

Remarks

None

Example usage

```
CASE @IDSIZE
    GetClientSize cp,l,t,w,h
    IF ControlExists(ps,IDHEADER)
        SetSize cp,0,0,w,20,IDPAGER
        pcRecalcSize(cp,IDPAGER)
    ENDIF
```

...

12.314pcGetPos

Syntax

INT = pcGetPos(win as WINDOW,id as UINT)

Description

Retrieves the current scroll position of the pager control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

The scroll position, in pixels.

Remarks

None

Example usage

```
pos = pcGetPos(demo, 500)
```

12.315pcGetButtonState

Syntax

INT = pcGetButtonState(win as WINDOW,id as UINT,button as INT)

Description

Retrieves the state of the specified button in a pager control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

button - Identifier of the button. 0 for Top/Left, 1 for Bottom/right

Return value

The return value can be one of the following:

@PGF_INVISIBLE - Scroll button is not visible

@PGF_NORMAL - Scroll button is in normal state

@PGF_GRAYED - Scroll button is in grayed state

@PGF_DEPRESSED - Scroll button is in depressed state

@PGF_HOT - Scroll button is in hot state

Remarks

None

Example usage

```
IF pcGetButtonState(cp, IDPAGER, 0) <> @PGF_INVISIBLE
...
ENDIF
```

12.316pcForwardMouse

Syntax

pcForwardMouse(win as WINDOW,id as UINT,bForward as INT)

Description

Enables or disables mouse forwarding for the pager control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

bForward - If this value is nonzero, mouse forwarding is enabled. If this value is zero, mouse forwarding is disabled.

Return value

None

Remarks

When mouse forwarding is enabled, the pager control forwards @IDMOUSEMOVE messages to the contained window.

Example usage

```
pcForwardMouse demo, 500, TRUE
```

12.317PLAYMIDI\$**Syntax**

POINTER = PLAYMIDI\$(strMidi as STRING,OPT bAsync as INT)

Description

Plays a stream of MIDI notes on the default MIDI device.

Parameters

strMidi - Music string containing notes to play.

bAsync - Optional. Specifies asynchronous playback if TRUE.

Return value

None if played synchronously or a pointer to the stream thread being played if asynchronous.

Remarks

Synchronous playback will wait until the entire stream is done playing. Asynchronous playback will return immediately and play the notes in the background using a separate execution thread.

Windows only allows one stream to open a device at a time. If a stream is already playing then attempting to start a second stream will fail.

See Also: [STOPMIDI\\$](#), [Music and Sound commands](#) for the format of the input string.

Example usage

```
PLAYMIDI$ "T180 N0 I0 O5 C8C#DD#EFF#GG#AA#B O6 CC#DD#EFF#GG#AA#B O7 C1"
```

12.318PLAYWAVE**Syntax**

INT = PLAYWAVE(name as ANYTYPE,flags as UINT)

Description

Plays a .WAV sound loaded from file or memory.

Parameters

name - String containing filename or a MEMORY variable with the loaded sound file.

flags - Playback flags.

Return value

.TRUE (1) if sound could be started or FALSE (0) if the sound hardware was busy.

Remarks

Valid values for flags are:

@SNDASYNC - The sound is played asynchronously and the function returns immediately after beginning the sound.

@SNDSYNC - The sound is played synchronously and the function does not return until the sound ends.

@SNDLOOP - The sound plays repeatedly. You must also specify @SNDASYNC

@SNDNOSTOP - If a sound is currently playing, the function immediately returns FALSE, without playing the requested sound.

To stop a sound use an empty string for the name or use NULL:

```
PLAYWAVE(NULL, 0)
```

Example usage

```
PLAYWAVE "c:\\media\\bark.wav", @SNDASYNC
```

12.319POWER

Syntax

```
DOUBLE = __POWER(base as DOUBLE,exponent as DOUBLE)
```

Description

Internal function used by the ^ operator.

Parameters

base - DOUBLE precision base

exponent - DOUBLE precision exponent

Return value

base ^ exponent

Remarks

Documented for sake of consistency. In all user code you should use the power operator ^.

Example usage

```
PRINT __POWER(2.0, 8.0)
```

12.320PRINT

Syntax

PRINT OPT win as WINDOW, ...

? OPT win as WINDOW, ...

Description

Outputs data to the console or a window.

Parameters

win - Optional window to print into.

... - Variable list of parameters to print.

Return value

None

Remarks

If *win* is omitted all output will be redirected to the console which must have been opened with the OPENCONSOLE command. Parameters may be variables, functions or literals. If a window is specified the text will be drawn at the position last set by the MOVE statement. A trailing comma in console mode leaves the caret at the end of the line.

For floating point types the SETPRECISION command determines the number of decimal places displayed

See Also: [SETPRECISION](#), [OPENCONSOLE](#)

Example usage

```
PRINT mywnd, A$, 2+2, "This is a test", LEFT$(A$, 10)
PRINT "This goes to the console",
```

12.321PRINTWINDOW

Syntax

PRINTWINDOW(win as WINDOW)

Description

Opens the standard printer dialog and sends the contents of the window to the selected printer. Output is properly scaled and sized for both portrait and landscape modes.

Parameters

win - Window to print

Return value

None

Remarks

For a window created with the @NOAUTODRAW style drawing in response to the @IDPAINT

message is redirected to the printer.

See Also: [OPENPRINTER](#), [PRTDIALOG](#)

Example usage

```
PRINTWINDOW win
```

12.322ProgressControl

Syntax

UINT = ProgressControl(win as WINDOW,l as INT,t as INT,w as INT,h as INT,flags as INT,exStyle as INT,id as UINT)

Description

Creates a progress bar control.

Parameters

win - Parent window or dialog containing the control.

l, t, w, h - Position and dimension of the control.

flags - Styles for the control.

exStyle - Extended window styles for the control

id - Control identifier.

Return value

If used with a window the return value is the handle of the control. If used with a dialog the return value is 0 and the handle to the control can be obtained with GETCONTROLHANDLE during the @IDINITDIALOG message

Remarks

The control can also be created using CONTROLEX with a window class of "msctls_progress32".

Styles

@PBS_SMOOTH - The progress bar displays progress status in a smooth scrolling bar instead of the default segmented bar. This style is supported only in the Windows Classic theme.

@PBS_VERTICAL - The progress bar displays progress status vertically, from bottom to top.

@PBS_MARQUEE - The progress indicator does not grow in size but instead moves repeatedly along the length of the bar, indicating activity without specifying what proportion of the progress is complete.

Example usage

```
ProgressControl cp,40,80,236,20,@BORDER|@PBS_SMOOTH,0,IDPROGRESSBAR
SetProgressRange cp,IDPROGRESSBAR,0,500
SetProgressStep cp,IDPROGRESSBAR,50
```

12.323ProgressStepIt

Syntax

ProgressStepIt(win as WINDOW,id as UINT)

Description

Advances the current position for a progress bar by the step increment and redraws the bar to reflect the new position.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

None

Remarks

When the position exceeds the maximum range value, this command resets the current position so that the progress indicator starts over again from the beginning.

Example usage

```
ProgressStepIt cp, IDPROGRESSBAR  
IF GetProgressPosition(cp, IDPROGRESSBAR) > 500 THEN SetProgressPosition(cp, IDPROGRESSBAR, 0)
```

12.324PROJECTGLOBAL

Syntax

PROJECTGLOBAL "option"

Description

Used to declare a group of variables as being globally visible.

Parameters

option - On or Off

Return value

None

Remarks

Variables are normally private to the source file they are defined/declared in. Declaring variables as GLOBAL allows other source file modules to use them when using the EXTERN keyword. EXTERN and GLOBAL are the heart of multi-module programming. PROJECTGLOBAL provides a means of creating multiple global variables in a project with a minimum of typing.

See Also [GLOBAL](#)

Example usage

```
'in the defining source file; "myglobals.iwb"
'file added to project just like all other source files
PROJECTGLOBAL "on"
  INT myvariable_1
  INT myvariable_2
  DIALOG mydialog_1
PROJECTGLOBAL "off"

-----
'in all other source files
'add this line at beginning of file
$INCLUDE "myglobals.iwb"
```

12.325PRTDIALOG

Syntax

STRING = PRTDIALOG(win as POINTER,vStart as INT BYREF,vEnd as INT BYREF,
vCopies as INT BYREF,vCollate as INT BYREF)

Description

Opens the standard system printer dialog and returns the name of the selected printer.

Parameters

win - [in] Parent window, dialog or NULL
vStart - [in][out] Starting page number
vEnd - [in][out] Ending page number
vCopies - [in][out] Number of copies requested
vCollate - [in][out] Indicates whether program performs collate or printer handles internally.

Return value

The name of the selected print device or an empty string "" if the user cancels the dialog.

Remarks

The function returns the name of the selected printer and modifies the integer variables *vStart*, *vEnd*, *vCopies* and *vCollate* to reflect the user's choices. Before using PRTDIALOG set the *vStart* and *vEnd* variables to indicate the number of pages available to print. If both are set to the same number then only the 'all pages' radio button will be available.

vCollate will be set to TRUE (1) only if the printer cannot handle collating and the user requests the function. *vCopies* will only be greater than one if the printer driver cannot handle multiple copies on its own.

See Also: [OPENPRINTER](#)

Example usage

```
DEF hPrt:UINT
DEF data:STRING
DEF name:STRING
DEF pagefrom, pageto, copies, collate:INT
pagefrom = 1
pageto = 1
copies = 1
name = PRTDIALOG(NULL, pagefrom, pageto, copies, collate)
hPrt = OPENPRINTER(name, "Test Document", "TEXT")
IF (hPrt)
  data = "This is a test of printing"
  data = data + chr$(13)
  data = data + "This is line 2"
  WRITEPRINTER hPrt, data
  CLOSEPRINTER hPrt
ENDIF
END
```

12.326PSET

Syntax

PSET(win as WINDOW, x as INT, y as INT, OPT clr as UINT)

Description

Sets the pixel in a window to the specified color, or the color of the current foreground pen.

Parameters

win - Window to set pixel.

x, y - Coordinates of pixel.

clr - Optional color. If not specified then the foreground color, set with FRONTPEN, is used.

Return value

None

Remarks

See also: [GETPIXEL](#)

Example usage

```
PSET mywnd, 10, 20, RGB(255,0,255)
```

12.327PushHeap

Syntax

PushHeap(lpMem as UINT)

Description

Pushes a heap pointer onto the heap stack. Used internally by string functions.

Parameters

lpMem - Heap pointer

Return value

None

Remarks

Only for use by command implementers. Do not call this function from high level code as severe memory corruption can occur.

See Also: [HeapClear](#)

Example usage

None

12.328PUT

Syntax

PUT(*vFile* as BFILE,*record* as INT,*var* as POINTER)

Description

.Puts one record to the random access binary file *vFile*. *record* must be greater than zero and the file must have been opened by OPENFILE. *var* can be any built-in or user defined variable type.

Parameters

vFile - Open binary file variable. Must have been opened with "W" or "A" mode.

record - Ones based record number.

var - Variable to store in file. Can be any built in type or UDT.

Return value

None

Remarks

See Also: [OPENFILE](#), [GET](#)

Example usage

```
PUT myfile, 20, phone_data
```

12.329RAND

Syntax

UINT = RAND(*num* as UINT,*opt max=0* as INT)

Description

Creates a pseudo random number.

Parameters

num - Beginning or ending number for the range.

max - Optional. If given then the random number will be between *num* and *max*. Otherwise random number will be between 0 and *num*.

Return value

An integer random number

Remarks

Maximum random range is $0 \leq \text{number} \leq 65535$

See Also [RND](#), [SEEDRND](#)

Example usage

```
num = RAND(50)
num2 = RAND(10,20)
```

12.330 RASTERMODE

Syntax

RASTERMODE(win as WINDOW, mode as INT)

Description

Sets the current raster mode for the window. The raster mode controls how source pixels from drawing operations are combined with the pixels already in the window. The default raster mode is @RMCOPYPEN

Parameters

win - Window to set the raster mode

mode - New mode

Return value

None

Remarks

See Also: RASTERMODE constants in the appendix.

Example usage

```
RASTERMODE mywnd, @RMXORPEN
```

12.331 rbAddBand

Syntax

rbAddBand(win as WINDOW, id as UINT, index as INT, style as UINT, cx as INT, cxMinChild as INT, cyMinChild as INT)

Description

Inserts a new band in the rebar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero-based index of the location where the band will be inserted.

If you set this parameter to -1, the control will add the new band at the last location.

style - Style of the band.

cx - Length of the band, in pixels.

cxMinChild - Minimum width of the child window, in pixels.

cyMinChild - Minimum height of the child window, in pixels.

Return value

None

Remarks

After adding a band associate a child control with [rbSetBandChild](#) or [rbSetBandChildHandle](#).

Styles

@RBBS_BREAK - Break to new line

@RBBS_FIXEDSIZE - Band cannot be sized

@RBBS_CHILDEDGE - Edge around top & bottom of child window

@RBBS_HIDDEN - Do not show

@RBBS_NOVERT - Do not show when vertical

@RBBS_FIXEDBMP - Bitmap does not move during band resize

@RBBS_VARIABLEHEIGHT - Allow autosizing of this child vertically

@RBBS_GRIPPERALWAYS - Always show the gripper

@RBBS_NOGRIPPER - Never show the gripper

Example usage

```
RebarControl cp,@RBS_VARHEIGHT,0,IDREBAR
rbAddBand cp,IDREBAR,0,@RBBS_CHILDEDGE,200,50,25
rbAddBand cp,IDREBAR,1,@RBBS_CHILDEDGE,250,50,40
rbSetBandText cp,IDREBAR,0,"Combo Box"
rbSetBandText cp,IDREBAR,1,"Toolbar"
```

12.332rbSetBandBitmap

Syntax

rbSetBandBitmap(win as WINDOW,id as UINT,index as INT,hBitmap as UINT)

Description

Sets the background bitmap used for a band.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero-based index of band.

hBitmap - Handle to the bitmap.

Return value

None

Remarks

None

Example usage

```
rbSetBandBitmap demo, 700, 2, hBitmap
```

12.333rbSetBandChild

Syntax

```
rbSetBandChild(win as WINDOW,id as UINT,index as int,child as UINT)
```

Description

Sets the child control contained by the band in a rebar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero-based index of band.

child - Child control identifier.

Return value

None.

Remarks

None.

Example usage

```
ComboBoxEx cp,0,0,100,100,@CTCOMBODROPDOWN|@VSCROLL,0,IDCOMBOBOX  
cbeAddString cp,IDCOMBOBOX,"String1"  
cbeAddString cp,IDCOMBOBOX,"String2"  
cbeAddString cp,IDCOMBOBOX,"String3"  
rbSetBandChild cp,IDREBAR,0,IDCOMBOBOX
```

12.334rbSetBandChildHandle

Syntax

```
rbSetBandChildHandle(win as WINDOW,id as UINT,index as int,hwndChild as UINT)
```

Description

Sets the child window contained by the band in a rebar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero-based index of band.

hwndChild - Child window identifier.

Return value

None

Remarks

None

Example usage

```
hChild = GetControlHandle(win,childID)
rbSetBandChildHandle win, 700, 0, hChild
```

12.335rbSetBandColors

Syntax

rbSetBandColors(win as WINDOW,id as UINT,index as INT,fore as UINT,back as UINT)

Description

Sets the foreground and background color of a band.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero-based index of band.

fore - Band foreground color.

back - Band background color.

Return value

None.

Remarks

None.

Example usage

```
rbSetBandColors demo, 700, 0, RGB(227,227,227), RGB(127,127,127)
```

12.336 rbSetBandText

Syntax

rbSetBandText(win as WINDOW, id as UINT, index as INT, text as STRING)

Description

Sets the text for a band in a rebar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero-based index of band.

text - New text for the band.

Return value

None.

Remarks

None.

Example usage

```
RebarControl cp, @RBS_VARHEIGHT, 0, IDREBAR
rbAddBand cp, IDREBAR, 0, @RBBS_CHILDEDGE, 200, 50, 25
rbAddBand cp, IDREBAR, 1, @RBBS_CHILDEDGE, 250, 50, 40
rbSetBandText cp, IDREBAR, 0, "Combo Box"
rbSetBandText cp, IDREBAR, 1, "Toolbar"
```

12.337 rbShowBand

Syntax

rbShowBand(win as WINDOW, id as UINT, index as int, bShow as int)

Description

Shows or hides a given band in a rebar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Zero-based index of band.

bShow - TRUE to show the band, FALSE to hide.

Return value

None.

Remarks

None.

Example usage

```
rbShowBand demo, 700, 1, FALSE
```

12.338READ

Syntax

```
INT = READ(vFile as FILE | BFILE, var as ANYTYPE)
```

Description

Reads data from a file opened with the OPENFILE command.

Parameters

vFile - FILE or BFILE successfully open for reading with OPENFILE

var - For ASCII files can be any built in type. For binary files (BFILE) can be any built in type or UDT.

Return value

Returns 0 on success or -1 if the file could not be read.

Remarks

In ASCII mode numeric data can be read sequentially if separated by space, tab, comma or new line characters. The WRITE statement separates numeric data with spaces.

String data can be any length. In ASCII mode READ will continue to read the string character by character until a new line or NULL terminator is found. Be sure to use a large enough string to accommodate the data. READ will overwrite string memory if the data is longer than the dimensioned string.

See Also: [OPENFILE](#), [WRITE](#), [GET](#), [PUT](#)

Example usage

```
READ myfile, nNumber
```

12.339READMEM

Syntax

```
READMEM(mPtr as MEMORY, record as INT, var as POINTER)
```

Description

Reads one record from allocated memory.

Parameters

mPtr - A MEMORY variable successfully initialized with ALLOCMEM or returned by an API function.

record - The ones based record number.

var - Variable to store memory into. Any built in or user defined type (UDT).

Return value

None

Remarks

See Also: [WRITEMEM](#), [ALLOCMEM](#), [FREEMEM](#)

Example usage

```
DEF buffer as MEMORY
DEF num as INT
ALLOCMEM buffer,100,LEN(num)
FOR x = 1 to 100
    WRITEMEM buffer,x,x
NEXT x

FOR x = 1 to 100
    READMEM buffer,x,num
    PRINT num
NEXT x
```

12.340 RebarControl

Syntax

UINT = RebarControl(win as WINDOW, flags as INT, exStyle as INT, id as UINT)

Description

Creates a rebar control.

Parameters

win - Parent window or dialog containing the control.

flags - Styles for the control.

exStyle - Extended window styles for the control

id - Control identifier.

Return value

If used with a window the return value is the handle of the control. If used with a dialog the return value is 0 and the handle to the control can be obtained with GETCONTROLHANDLE during the @IDINITDIALOG message

Remarks

The control can also be created using CONTROLEX with a window class of "ReBarWindow32". Rebar controls act as containers for child windows. An application assigns child windows, which are often other controls, to a rebar control band. Rebar controls contain one or more bands, and each band can have any combination of a gripper bar, a bitmap, a text label, and a child window. However, bands cannot contain more than one child window.

Styles

@RBS_TOOLTIPS - Not yet supported.

@RBS_VARHEIGHT - The rebar control displays bands at the minimum required height, when possible.

@RBS_BANDBORDERS - The rebar control displays narrow lines to separate adjacent bands.

@RBS_FIXEDORDER - The rebar control always displays bands in the same order.

@RBS_REGISTERDROP - The rebar control generates @RBN_GETOBJECT notification messages when an object is dragged over a band in the control.

@RBS_AUTOSIZE - The rebar control will automatically change the layout of the bands when the size or position of the control changes. An @RBN_AUTOSIZE notification will be sent when this occurs.

@RBS_VERTICALGRIPPER - This always has the vertical gripper (default for horizontal mode)

@RBS_DBLCLKTOGGLE - The rebar band will toggle its maximized or minimized state when the user double-clicks the band.

Example usage

```
RebarControl cp,@RBS_VARHEIGHT,0,IDREBAR
rbAddBand cp,IDREBAR,0,@RBBS_CHILDEDGE,200,50,25
rbAddBand cp,IDREBAR,1,@RBBS_CHILDEDGE,250,50,40
rbSetBandText cp,IDREBAR,0,"Combo Box"
rbSetBandText cp,IDREBAR,1,"Toolbar"
```

12.341RECT

Syntax

RECT(win as WINDOW,l as INT,t as INT,w as INT,h as INT,OPT outline as UINT, OPT fill as UINT)

Description

Draws a rectangle in the window.

Parameters

win - Window to draw into.

l, *t*, *w*, *h* - Coordinates and dimensions of the rectangle.

outline - Optional outline color.

fill - Optional fill color.

Return value

None.

Remarks

If *outline* and *fill* are not specified the rectangle will be drawn in the current foreground color.

Example usage

```
RECT mywindow, 0,0,100,100, RGB(255,0,0), RGB(0,0,255)
```

12.342 REDRAWFRAME

Syntax

REDRAWFRAME(win as WINDOW, OPT id=0 as INT)

Description

Redraws the border, caption and non-client areas of a window, dialog or control.

Parameters

win - WINDOW or DIALOG.

id - Optional control ID.

Return value

None

Remarks

If *id* is specified then the frame of a control is changed whose parent is the window or dialog specified. Used after modifying the style of a window, dialog or control.

See Also: [MODIFYSTYLE](#), [MODIFYEXSTYLE](#).

Example usage

```
' Remove the 3D border from an edit control
CONST WS_EX_CLIENTEDGE = 0x200
MODIFYEXSTYLE mywin, 0, WS_EX_CLIENTEDGE, 2
REDRAWFRAME mywin, 2
```

12.343 REGGETDWORD

Syntax

string = REGGETDWORD(string key, pointer ValueName, opt defValue as UINT)

Description

Returns a registry dword value.

Parameters

key - The registry key path.

ValueName - The key to retrieve.

defValue - Optional. Value to return if key doesn't exist.

Return value

The dword value contained in the key.

Remarks

setting ValueName to NULL will retrieve the default value

Example usage

```
nValue=RegGetDword("HKEY_CURRENT_USER\\Software\\XYZCorp\\File","OpenLast",TRUE)
```

12.344REGGETSTRING

Syntax

string = REGGETSTRING(string key, pointer ValueName, opt defValue as string)

Description

Returns a registry string value.

Parameters

key - The registry key path.

ValueName - The key to retrieve.

defValue - Optional. Value to return if key doesn't exist.

Return value

The string value contained in the key.

Remarks

setting ValueName to NULL will retrieve the default value

Example usage

```
str=RegGetString("HKEY_CURRENT_USER\\Software\\XYZCorp\\File","User Name","Unregisterc
```

12.345REGSETDWORD

Syntax

error = REGSETDWORD(string key, pointer ValueName, Value as UINT)

Description

Sets a registry dword value.

Parameters

key - The registry key path.

ValueName - The key to set.

Value - The value to set.

Return value

Returns 0 for no error, or 1 if the value could not be set.

Remarks

Setting ValueName to NULL will set the default value. Will create the key if it doesn't exist.

Example usage

```
Result=RegSetDword("HKEY_CURRENT_USER\\Software\\XYZCorp\\File","UseColor",1)
```

12.346 REGSETSTRING

Syntax

error = REGSETSTRING(string key, pointer ValueName, Value as string)

Description

Sets a registry string value.

Parameters

key - The registry key path.

ValueName - The key to set.

Value - The value to set.

Return value

Returns 0 for no error, or 1 if the value could not be set.

Remarks

Setting ValueName to NULL will set the default value. Will create the key if it doesn't exist.

Example usage

```
Result=RegSetString("HKEY_CURRENT_USER\\Software\\XYZCorp\\File","LastOpen","blah.txt")
```

12.347 RELEASEHDC

Syntax

RELEASEHDC(win as WINDOW, hdc as UINT, OPT bRedraw=1 as INT)

Description

Releases the device context previously acquired with GETHDC.

Parameters

win - Window the device context belongs to.

hdc - Handle to the device context returned by GETHDC.

bRedraw - Optional. If 0 then the window is not redrawn after release. Default is to redraw and show changes.

Return value

None

Remarks

Every call to GETHDC must be matched in pairs to RELEASEHDC.

See Also: [GETHDC](#)

Example usage

```
hdc = GETHDC(win)
...
RELEASEHDC(win, hdc)
```

12.348REMOVEDIR

Syntax

INT = REMOVEDIR(path as STRING)

Description

Removes a directory

Parameters

path - Full path to directory to be removed

Return value

0 on failure or greater than 0 if successful

Remarks

Directory must be empty before it can be removed. The path should not contain a trailing slash.

See Also: [CREATEDIR](#), [DELETEDFILE](#)

Example usage

```
REMOVEDIR "c:\\myfiles\\temp"
```

12.349REMOVEMENUITEM

Syntax

REMOVEMENUITEM(win as WINDOW, pos as UINT, id as UINT)

Description

Removes a menu item. If id=0 then the entire menu specified by position is removed.

Parameters

win - Window or dialog containing the menu.

pos - Zero based location of the pull down menu.

id - Identifier of the menu item to be removed.

Return value

None

Remarks

See Also: [ADDMENUITEM](#)

Example usage

```
REMOVEMENUITEM mywnd, 0, 1
```

12.350 REPLACE\$

Syntax

REPLACE\$(dest as STRING, start as INT, count as INT, source as STRING)

Description

Replaces characters in one string with one or more characters from another.

Parameters

dest - String with characters to be replaced.

start - Ones based starting position of the replacement.

count - Number of characters to replace.

source - String containing the replacement characters.

Return value

None

Remarks

If *count* specifies more characters than exists in *source* then all of the characters in *source* are copied to *dest* up to the total length of the destination string.

Example usage

```
DEF s:string  
s = "All good DOGS go to heaven"  
REPLACE$ s,10,3,"dog"  
PRINT s
```

12.351 RESTORE

Syntax

RESTORE(blockname as DATABLOCK BYREF)

Description

Restores the data pointer in the block to the beginning.

Parameters

blockname - Name of the data block specified in the DATABEGIN statement.

Return value

None

Remarks

See Also: [DATABEGIN](#), [DATAEND](#), [DATA](#), [GETDATA](#)

Example usage

```
DEF a as INT

RESTORE mydata

FOR x = 1 to 6
  GETDATA mydata,a
  PRINT a
NEXT x

DO:UNTIL INKEY$ <> ""
END

DATABEGIN mydata
DATA 1,2,3
DATA 4,10,2002
DATAEND
```

12.352RETURN

Syntax

RETURN OPT value

Description

Returns from a subroutine.

Parameters

value - An optional value to return to the caller. There is automatic string conversion (from/to unicode) when returning a string from a function:

Return value

None

Remarks

You can return any built in variable type or UDT. Strings and UDT's are dynamically copied and returned. This assures that any local string or UDT can safely be used in the RETURN statement.

The return value is optional only if your subroutine wasn't declared as having one.

See Also: [SUB](#)

Example usage

```
PRINT mysub(2,5)
DO:UNTIL INKEY$ <> ""
END

SUB mysub(c as INT, a as INT), INT
  i = c * a
RETURN i
ENDSUB
```

```
'automatic string conversion (from/to unicode)
$define UNICODE
$define WIN32_LEAN_AND_MEAN
#include "windowssdk.inc"

MessageBoxW(0, function1(), L"ansi->unicode")
MessageBoxA(0, function2(), "unicode->ansi")

sub function1(),wstring
return "hello" ' will be converted to unicode, using the current codepage
endsub

sub function2(),string
return L"hello" ' will be converted to ansi, using the current codepage
endsub
```

12.353RGB

Syntax

UINT = RGB(r as INT, g as INT, b as INT)

Description

Creates a combined RGB color from separate components.

Parameters

r - Red component of the color
g - Green component of the color
b - Blue component of the color

Return value

A combined color value

Remarks

Component value range is from 0 to 255. Most drawing commands that accept a color value use the RGB format.

Example usage

```
FRONTPEN mywin, RGB(0,0,255)
```

12.354RGNFROMBITMAP

Syntax

UINT = RGNFROMBITMAP(id as ANYTYPE,OPT cTransparentColor = 0 as UINT, opt
cTolerance = 0x101010 as UINT)

Description

.Creates a window region from a bitmap

Parameters

id - File name or resource ID of the bitmap.

cTransparentColor - The color used for the regions mask.

cTolerance - If specified creates a range of transparent colors.

Return value

A handle to a region (hrgn).

Remarks

RGNFROMBITMAP creates a non-rectangular region by using the color range from *cTransparentColor* to *cTolerance* as a transparency color. The region handle returned can be applied as a windows region or used with the SETBUTTONRGN command to create non-rectangular button control.

Example usage

```
hrgn = RGNFROMBITMAP (GETSTARTPATH+"rgn_bmp1.bmp")
SETCONTROLCOLOR d1,BUTTON_1,RGB(255,255,255),RGB(10,100,128)
SETBUTTONRGN d1,BUTTON_1, hrgn
SETHTCOLOR d1,BUTTON_1,RGB(20,138,138)
SETBUTTONBORDER d1,BUTTON_1,0
```

12.355RIGHT\$ / WRIGHT\$

Syntax

STRING = RIGHT\$(str as STRING,count as INT)

WSTRING = WRIGHT\$(str as WSTRING,count as INT)

Description

Extracts the last (rightmost) characters of a string and returns a copy of the extracted string.

Parameters

str - The string to extract characters from.

count - Leftmost count of characters to extract.

Return value

A copy of the extracted string.

Remarks

Count can be zero in which case an empty string is returned.

See Also: [LEFTS](#), [MIDS](#)

Example usage

```
A$ = "A cat and dog"
IF RIGHT$(A$,3) = "dog" THEN PRINT "bow wow"
```

12.356RND

Syntax

FLOAT = RND(num as FLOAT ,opt max=0 as FLOAT)

Description

Creates a pseudo random number.

Parameters

num - Beginning or ending number for the range.

max - Optional. If given then the random number will be between *num* and *max*. Otherwise random number will be between 0 and *num*.

Return value

A floating point random number

Remarks

Maximum random range is $-32765.0 \leq \text{number} \leq 32766.0$

See Also [RAND](#), [SEEDRND](#)

Example usage

```
num = RND(1.0)
num2 = RND(10.0, 20.5)
```

12.357RTRIM\$ / WRTRIM\$

Syntax

STRING = RTRIM\$(str as STRING)

WSTRING = WRTRIM\$(str as WSTRING)

Description

Trims trailing whitespace characters from the string.

Parameters

str - The input string

Return value

A copy of the input string minus any trailing whitespace characters.

Remarks

Whitespace as defined by this function includes space and tab characters.

See Also: [LTRIM\\$](#)

Example usage

```
PRINT RTRIM$("Remove trailing spaces      ")
```

12.358S2W

Syntax

WSTRING = S2W(in as STRING)

Description

Converts an ANSI string to a wide character (Unicode) string .

Parameters

in - String to convert..

Return value

The converted string..

Remarks

None

See Also: [W2S](#)

Example usage

```
w = S2W("A string to convert")
```

12.359SCHAR

Syntax

schar = SCHAR(exp)

Description

Converts a value to a single signed char.

Parameters

exp - A numeric expression

Return value

A single signed character or value (-128 to 127 inclusive)

Remarks

.

Example usage

```
print SCHAR(65)
'prints 65
print chr$(SCHAR(65))
'prints A

print SCHAR(-65)
'prints -65
print SCHAR(-191)
```

```
'prints 65
```

12.360SEEDRND

Syntax

SEEDRND(num as INT)

Description

Seeds the random number generator with a specific value.

Parameters

num - The random seed.

Return value

None

Remarks

On program execution the random number generator seed is set to the current Windows tick count guaranteeing that a repeating random sequence is unlikely to occur. If you use the same seed every time then the random sequence will be identical on successive runs.

See Also: [RND](#), [RAND](#)

Example usage

```
SEEDRND(3433334)
```

12.361SEEK

Syntax

INT64 = SEEK(*fptr* as BFILE, *OPT position* as INT64)

Description

Moves the file pointer of a binary file to the specified position or returns the current position.

Parameters

fprt - A variable of type BFILE successfully opened with OPENFILE

position - Optional. Position to set file pointer to.

Return value

If position is omitted then this function returns the current file position.

Remarks

See Also [OPENFILE](#)

Example usage

```
where = SEEK(myfile)  
SEEK myfile, where+2
```

12.362SELECT

Syntax

SELECT value

Description

Sets the test value for multiple CASE statements. Each CASE statement will contain a condition to evaluate. Must be followed by an ENDSELECT statement.

Parameters

value - variable or condition to compare with each CASE statement

Return value

None

Remarks

See Also: [CASE](#), [CASE&](#), [DEFAULT](#), [ENDSELECT](#)

Example usage

```
A = 1
SELECT A
    CASE 1
        PRINT "TRUE!"
    CASE 2
        PRINT "You wont see this text"
    DEFAULT
        PRINT "None of the above"
ENDSELECT
```

12.363SENDMESSAGE

Syntax

UINT = SENDMESSAGE(win as ANYTYPE, msg as UINT, wparam as UINT, lparam as ANYTYPE, OPT id=0 as UINT)

Description

Sends a message to a dialog, window or control.

Parameters

win - Window, dialog or HWND.

msg - Message to send.

wparam - Numeric parameter.

lparam - Any type of variable, constant or UDT.

id - Optional control identifier.

Return value

Message dependent.

Remarks

Convenient replacement for the SendMessageA Windows API.

Example usage

```
SENDMESSAGE mywnd, WM_CLOSE, 0, 0
```

See the sample file dirselector.iwb for a complete example of using SENDMESSAGE

12.364SEPARATOR

Syntax

SEPARATOR

Description

Inserts a separator bar into a menu definition.

Parameters

None

Return value

None

Remarks

See Also: [BEGINMENU](#), [ENDMENU](#), [MENUITEM](#), [MENUTITLE](#), [BEGINPOPUP](#), [ENDPOPUP](#), [BEGININSERTMENU](#)

Example usage

```
BEGINMENU win
    MENUTITLE "&File"
    MENUITEM "Open", 0, 1
    MENUITEM "Close", 0, 2
    BEGINPOPUP "Save As..."
        MENUITEM "Ascii", 0, 3
        MENUITEM "Binary", 0, 4
    ENDPOPUP
    SEPARATOR
    MENUITEM "&QUIT", 0, 5
    MENUTITLE "&Edit"
    MENUITEM "Cut", 0, 6
ENDMENU
```

12.365SET_INTERFACE

Syntax

SET_INTERFACE object as COMREF, name

Description

Assigns an interface to a COM reference.

Parameters

object - A variable of type COMREF

name - The name as defined in the INTERFACE statement

Return value

None

Remarks

An COM object cannot be used until its interface is set. Once an interface is assigned, and the object initialized with CoCreateInstance or another dedicated function, you can begin using methods in the object.

See Also: [INTERFACE](#), [ENDINTERFACE](#), [STDMETHOD](#), [DEFINE_GUID](#)

Example usage

```
DEF lpTest,lpDD7 as COMREF
DEF freq as UINT
DEF scanline as UINT
lpTest = 0
SET_INTERFACE lpTest, IDirectDraw
SET_INTERFACE lpDD7, IDirectDraw7
DirectDrawCreate(0,lpTest,0)

if(lpTest <> 0)
    lpTest->QueryInterface(IID_IDirectDraw7,lpDD7)
    lpTest->GetMonitorFrequency(freq)
    lpTest->GetScanLine(scanline)
    PRINT "Frequency: ",freq
    PRINT "Scanline: ",scanline
    lpTest->Release()
    if lpDD7 <> 0
        lpDD7->Release()
    endif
endif
```

12.366 SETBUTTONBITMAPS

Syntax

SETBUTTONBITMAPS(win as window,id as int,hNormal as UINT,hHot as UINT,hSelected as UINT)

Description

Sets the bitmaps used by a @RGNBUTTON.

Parameters

hNormal - Bitmap displayed as default by the button.

hHot - Bitmap displayed during hot tracking of the button.

hSelected - Reserved for future use, should be NULL

Return value

None

Remarks

The bitmaps should be created the same dimensions as the button control. This command should only be used on a `@RGNBUTTON` control or you may have resource leaks.

Once the bitmaps are set the control owns the handles and will free them automatically when the button control is destroyed.

Example usage

```
SETCONTROLCOLOR d1,BUTTON_4,RGB(200,200,200),RGB(10,100,128)
SETBUTTONBITMAPS d1,BUTTON_4,LOADIMAGE(GETSTARTPATH+"button_bmp_normal2.bmp"),@IMGBITMAP
LOADIMAGE(GETSTARTPATH+"button_bmp_hot2.bmp"),@IMGBITMAP,0
```

12.367SETBUTTONBORDER

Syntax

SETBUTTONBORDER(win as WINDOW,id as INT,width as int)

Description

Sets the width of a `@RGNBUTTON` border.

Parameters

win - Parent window or dialog containing the control.

id - Controls identifier.

width - Width of the buttons border.

Return value

None

Remarks

The allowed values for the width parameter depends on the type of `@RGNBUTTON` used:

-For a button with a region and 3D border (default) the width value can be 0, 1 or 2. 0 turns the border off

-For a button with a region and `@CTLBTNFLAT` style the width value can be 0 to n. 0 turns the border off, n can be any size.

-For a button without a region the width value can be 0 or 1. In other words On or Off

Example usage

```
'button 3 is a regioned button with automatic hot tracking color
SETCONTROLCOLOR d1,BUTTON_3,RGB(255,255,255),RGB(10,100,128)
SETBUTTONRGN d1,BUTTON_3,RGNFROMBITMAP(GETSTARTPATH+"rgn_bmp2.bmp")
SETHTCOLOR d1,BUTTON_3,RGB(20,138,138)
SETBUTTONBORDER d1,BUTTON_3,2
```

12.368SETBUTTONRGN

Syntax

SETBUTTONRGN(win as window,id as int,hrgn as UINT)

Description

.Sets the display region used by a button of type @RGNBUTTON

Parameters

win - WINDOW or DIALOG parent of control

id - Controls identifier

hrgn - Handle to a region created by RGNFROMBITMAP.

Return value

None

Remarks

The region passed to this command becomes the property of the control. No further action should be done with the regions handle. If you wish to share a region between multiple control used the [COPYRGN](#) command.

Example usage

```
hrgn = RGNFROMBITMAP(GETSTARTPATH+"rgn_bmp1.bmp")
SETCONTROLCOLOR d1,BUTTON_1,RGB(255,255,255),RGB(10,100,128)
SETBUTTONRGN d1,BUTTON_1,COPYRGN(hrgn)
SETHTCOLOR d1,BUTTON_1,RGB(20,138,138)
SETBUTTONBORDER d1,BUTTON_1,0
```

12.369SETCAPTION

Syntax

SETCAPTION(win as WINDOW,text as STRING)

Description

Sets the title bar caption of a window or dialog.

Parameters

win - Window or dialog.

text - New caption text.

Return value

None

Remarks

See Also: [GETCAPTION](#)

Example usage

```
SETCAPTION mywnd, "New text"
```

12.370 SETCOMPROPERTY

Syntax

```
INT = SetComProperty(IDispatch obj, STRING property,...)
```

Description

Sets the property of a COM object using script syntax.

Parameters

obj - The Object returned by the CREATECOMOBJECT command.

property - Name of the property to set.

... - Optional parameter list.

Return value

0 for Success.

Remarks

Parameters strings are specified by using C printf-like specifiers. The table below shows the supported specifiers:

Identifier	Type
%d	INT
%u	UINT
%e	DOUBLE
%b	INT
%v	VARIANT UDT.
%B	BSTR - Created with the SysAllocString API function
%s	STRING
%S	WSTRING
%T	WSTRING
%o	IDispatch COM object
%O	IUnknown COM object
%t	C time_t UDT
%W	SYSTEMTIME UDT
%f	FILETIME UDT
%D	C date type.
%p	POINTER
%m	Specifies a missing/optional argument

Example usage(s)

```
SetComProperty(_xmlDoc, ".Async = %b", FALSE)
```

```
SetComProperty(xlCells, ".Interior.Color = %d", RGB(0xee,0xdd,0x82))
SetComProperty(xlCells, ".Interior.Pattern = %d", 1) /* xlSolid */
SetComProperty(xlCells, ".Font.Size = %d", 13)
SetComProperty(xlCells, ".Borders.Color = %d", RGB(0,0,0))
SetComProperty(xlCells, ".Borders.LineStyle = %d", 1) /* xlContinuous */
SetComProperty(xlCells, ".Borders.Weight = %d", 2) /* xlThin */
```

12.371SETCONTROLCOLOR

Syntax

SETCONTROLCOLOR(win as WINDOW,id as INT,fg as UINT,bg as UINT)

Description

Changes the color of a control.

Parameters

win - Window or dialog containing control.

id - Identifier of control.

fg - New foreground color.

bg - New background color.

Return value

None.

Remarks

The foreground color is used for the text color in most controls. @SYSBUTTON controls do not support color changes. Rich Edit control have their own text coloring functions. Checkbox and radio button controls with the style BS_PUSHLIKE do not support color changes.

See Also: [CONTROL](#)

Example usage

```
SETCONTROLCOLOR mydlg, 7, RGB(255,255,255), RGB(0,0,0)
```

12.372SETCONTROLNOTIFY

Syntax

SETCONTROLNOTIFY win as WINDOW,id as INT,bTab as INT,bEnter as INT

Description

Turns control notification of enter and tab keys on or off.

Parameters

win - Window or dialog containing the control.

id - Control identifier.

bTab - TRUE to send @ENTABKEY notifications, FALSE otherwise.

bEnter - TRUE to send @ENENTERKEY notifications, FALSE otherwise.

Return value

None

Remarks

Designed for edit controls placed in a Window but will work with other controls as well.

@ENTABKEY and @ENENTERKEY are special notification codes sent in @NOTIFYCODE when enabled. By default both are disabled for compatibility with older code and controls used in a dialog where the enter and TAB keys have a different meaning.

Example usage

```
WINDOW win
OPENWINDOW win,0,0,295,199,0,0,"Caption",&handler
CONTROL win,@EDIT,"Press Enter or TAB",83,39,124,27,0x50800000|@CTEDITAUTOH,5
SETFONT win,"Ariel",9,400,0,5
SETCONTROLNOTIFY(win,5,1,1)
SetFocus win,5
WAITUNTIL win=0
END
.
SUB handler( ),INT
  SELECT @MESSAGE
    CASE @IDCONTROL
      IF @CONTROLID = 5
        SELECT @NOTIFYCODE
          CASE @ENENTERKEY
            MESSAGEBOX win,GetControlText(win,5),"Enter Pressed!"
            SetFocus win,5
          CASE @ENTABKEY
            MESSAGEBOX win,"Tab key pressed","Info"
            SetFocus win,5
          ENDSELECT
        ENDIF
      CASE @IDCREATE
        CENTERWINDOW win
      CASE @IDCLOSEWINDOW
        CLOSEWINDOW win
    ENDSELECT
  RETURN 0
ENDSUB
```

12.373SETCONTROLTEXT

Syntax

SETCONTROLTEXT(win as WINDOW,id as UINT,text as STRING)

Description

Sets the text of a control. Equivalent to the WM_SETTEXT windows message.

Parameters

win - Window or dialog containing the control.
id - Identifier of control.
text - New text for control.

Return value

None

Remarks

Not all controls respond to the SETCONTROLTEXT function. For edit controls SETCONTROLTEXT will set the text in the edit control. For all other controls it will retrieve the caption text of the control.

See Also: [GETCONTROLTEXT](#)

Example usage

```
SETCONTROLTEXT mydlg, 17, "Cancel"
```

12.374SETCURSOR

Syntax

SETCURSOR(win as WINDOW, type as INT, OPT handle=0 as UINT)

Description

Changes the currently displayed cursor in a window or dialog.

Parameters

win - Window or dialog

type - @CSARROW, @CSWAIT or @CSCUSTOM

handle - Optional. For @CSCUSTOM this is the handle returned by LOADIMAGE

Return value

None

Remarks

See Also: [SETICON](#), [LOADIMAGE](#)

Example usage

```
SETCURSOR mywnd, @CSWAIT
```

12.375SETEXITCODE

Syntax

SETEXITCODE(nCode as INT)

Description

Sets the exit code for your process.

Parameters

nCode - Exit code to return to windows.

Return value

None.

Remarks

The default exit code for a process is 0. Use this statement before END to report a different code. Console programs sometimes use this to indicate an error by returning -1.

Example usage

```
SETEXITCODE -1
```

12.376 SETFOCUS

Syntax

```
SETFOCUS(win as WINDOW,OPT id = 0 as UINT)
```

Description

Sets the input focus to a window, dialog or control.

Parameters

win - Window or dialog.

id - Optional control identifier.

Return value

None.

Remarks

The SETFOCUS command sends a WM_KILLFOCUS message to the window that loses the keyboard focus and a WM_SETFOCUS message to the window that receives the keyboard focus. It also activates either the window that receives the focus or the parent of the window that receives the focus.

Example usage

```
SETFOCUS win1, 17
```

12.377 SETFONT

Syntax

```
SETFONT(win as WINDOW,typeface as STRING,height as INT,weight as INT,OPT flags=0 as  
UINT,OPT id=0 as UINT)
```

Description

Changes the font of a window or control.

Parameters

win - Window or parent dialog.

typeface - Name of new font.

height - Size of font in points.

weight - Weight of font.

flags - Style flags for font and character set.

id - Optional control identifier.

Return value

None

Remarks

height and *weight* can both be 0 in which case a default size and weight will be used. Weight ranges from 0 to 1000 with 700 being standard for bold fonts and 400 for normal fonts. Flags can be a combination of @SFITALIC, @SFUNDERLINE, or @SFSTRIKEOUT for italicized, underlined, and strikeout fonts. If an *id* is specified then the font of a control in the window or dialog is changed.

The height parameter is specified in points. 1 point is equal to 1/72nd of an inch. If you want a font that is 1/2 an inch high you would specify a point size of 36.

Certain fonts may have more than one character set. Normally this information is set automatically by the flag value returned by FONTREQUEST. You can set the character set manually by using the following values OR'ed in with the *flags*

```
ANSI_CHARSET = 0
DEFAULT_CHARSET = 0x00010000
SYMBOL_CHARSET = 0x00020000
SHIFTJIS_CHARSET = 0x00800000
HANGEUL_CHARSET = 0x00810000
GB2312_CHARSET = 0x00860000
CHINESEBIG5_CHARSET = 0x00880000
OEM_CHARSET = 0x00FF0000
JOHAB_CHARSET = 0x00820000
HEBREW_CHARSET = 0x00B10000
ARABIC_CHARSET = 0x00B20000
GREEK_CHARSET = 0x00A10000
TURKISH_CHARSET = 0x00A20000
VIETNAMESE_CHARSET = 0x00A30000
THAI_CHARSET = 0x00DE0000
EASTEUROPE_CHARSET = 0x00EE0000
RUSSIAN_CHARSET = 0x00CC0000
MAC_CHARSET = 0x004D0000
```

BALTIC_CHARSET = 0x00BA0000

See Also: [FONTREQUEST](#)

Example usage

```
SETFONT mywin, "Terminal", 20, 700, @SFITALIC | 0x00FF0000
```

12.378SETHORIZEXTENT

Syntax

SETHORIZEXTENT(win as WINDOW,id as INT,width as INT)

Description

Sets the amount a list box or the list box portion of a combo box can be scrolled horizontally.

Parameters

win - Window or dialog containing the list box or combo box.

id - Identifier of control.

width - Width in pixels

Return value

None

Remarks

Control must have been created with the @HSCROLL style.

Example usage

```
SETHORIZEXTENT mywnd, 1, 100
```

12.379SETHTCOLOR

Syntax

SETHTCOLOR(win as WINDOW,id as INT,hc as UINT)

Description

.Sets the hot tracking color used by a button of type @RGNBUTTON

Parameters

win - Parent window or dialog of the control.

id - Controls identifier.

hc - The hot tracking color.

Return value

NONE

Remarks

A region button supports an automatic hot tracking color. The hot tracking color is displayed whenever the mouse is over the buttons client area. If the region button has a normal bitmap, but no hot tracking bitmap, then this color is used to tint the normal bitmap to provide the hot tracking effect. If the region button has both a normal and hot tracking bitmap then this color is ignored. Under all other circumstances the color is displayed as a solid color.

Example usage

```
button 3 is a regioned button with automatic hot tracking color
SETCONTROLCOLOR d1,BUTTON_3,RGB(255,255,255),RGB(10,100,128)
SETBUTTONRGN d1,BUTTON_3,RGNFROMBITMAP(GETSTARTPATH+"rgn_bmp2.bmp")
SEHTCOLOR d1,BUTTON_3,RGB(20,138,138)
SETBUTTONBORDER d1,BUTTON_3,2
```

12.380SETICON

Syntax

SETICON(win as WINDOW,handle as UINT)

Description

Sets the icon for the window or dialog.

Parameters

win - Window or dialog.

handle - Handle of new icon returned by the LOADIMAGE function.

Return value

None

Remarks

See Also: [LOADIMAGE](#)

Example usage

```
hIcon = LOADIMAGE("c:\\images\\prog.ico", @IMGICON)
SETICON mywindow, hIcon
```

12.381SETID

Syntax

SETID "name", value

Description

Creates an @ constant.

Parameters

name - Name of the constant. Must be unique.

value - UINT value of constant

Return value

None

Remarks

@ constants are used throughout the IWBASIC language and are specially syntax colored in the IDE. Unlike the CONST statement a SETID value cannot contain any math operators. The constant is evaluated at compile time.

See Also: [CONST](#)

Example usage

```
SETID "MYMESSAGE", 0x400
SELECT @CLASS
    CASE @MYMESSAGE
    ...
```

12.382SETLBCOLWIDTH

Syntax

SETLBCOLWIDTH(win as WINDOW, id as INT, width as INT)

Description

Sets the width in pixels of columns in a multi-column list box. A multi-column list box has fixed column widths.

Parameters

win - Window or dialog containing the control.

id - Identifier of the control.

width - New column width in pixels.

Return value

None

Remarks

None

Example usage

```
SETLBCOLWIDTH win, 7, 50
```

12.383SETLINESTYLE

Syntax

SETLINESTYLE(win as WINDOW, style as INT, width as INT)

Description

Sets the line drawing style and line width for a window.

Parameters

win - Window to set style.
style - New line drawing style.
width - New line width.

Return value

None

Remarks

Style can be one of @LSSOLID, @LSDASH, @LSDOT, @LSDASHDOT, @LSDASHDOTDOT or @LSINSIDE. Width determines the line width for solid lines. Note: Only @LSSOLID is valid with line widths greater than 1.

See Also: [LINE](#), [LINETO](#)

Example usage

```
SETLINESTYLE win1, @LSSOLID, 4
```

12.384SETMENU

Syntax

SETMENU(*win* as WINDOW,*hmenu* as UINT)

Description

Replaces the menu bar in a window or dialog. Used internally by the menu creation macros.

Parameters

win - Window or dialog
hmenu - New menu returned by CREATEMENU

Return value

None

Remarks

See Also: [CREATEMENU](#)

Example usage

```
SETMENU mywin, hmenu
```

12.385SETPRECISION

Syntax

SETPRECISION (*places* as INT)

Description

Sets the output display precision for the PRINT and STR\$ commands

Parameters

places - Number of decimal places to display or include in conversion.

Return value

None

Remarks

By default the PRINT statement is set to display 2 decimal places of FLOAT and DOUBLE types. The SETPRECISION command only effects the display or conversion to a string of floating point numbers and not the internal storage of the number. PRINT and STR\$ support a maximum of 40 digits after the decimal point.

See Also: [PRINT](#), [STR\\$](#)

Example usage

```
SETPRECISION 10  
PRINT 1.2234334556
```

12.386SetProgressBarColor

Syntax

SetProgressBarColor(win as WINDOW,id as UINT,clr as UINT)

Description

Sets the color of the progress indicator bar in the progress bar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

clr - The color of the progress bar.

Return value

None

Remarks

None

Example usage

```
SetProgressBarColor cp, IDPROGRESS, RGB(255,0,0)
```

12.387SetProgressDelta

Syntax

SetProgressDelta(win as WINDOW,id as UINT,delta as INT)

Description

Advances the current position of a progress bar by a specified increment and redraws the bar to reflect the new position.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

delta - The increment.

Return value

None.

Remarks

If the increment results in a value outside the range of the control, the position is set to the nearest boundary. The behavior of this command is undefined if it is sent to a control that has the @PBS_MARQUEE style.

Example usage

```
SetProgressDelta demo, 600, 10
```

12.388SetProgressMarquee

Syntax

```
SetProgressMarquee(win as WINDOW,id as UINT,bEnable as INT,time as INT)
```

Description

Sets the progress bar to marquee mode. This causes the progress bar to move like a marquee.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

bEnable - Indicates whether to turn the marquee mode on or off.

time - Time, in milliseconds, between marquee animation updates. If this parameter is zero, the marquee animation is updated every 30 milliseconds.

Return value

None.

Remarks

Use this command when you do not know the amount of progress toward completion but wish to indicate that progress is being made.

Example usage

```
SetProgressMarquee demo, 600, TRUE, 250
```

12.389SetProgressPosition

Syntax

SetProgressPosition(win as WINDOW,id as UINT,pos as INT)

Description

Sets the current position for a progress bar and redraws the bar to reflect the new position.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

pos - New position for the indicator.

Return value

None

Remarks

If *pos* is outside the range of the control, the position is set to the closest boundary. This command does not work with marquee style progress bars.

Example usage

```
ProgressStepIt cp, IDPROGRESSBAR  
IF GetProgressPosition(cp, IDPROGRESSBAR) > 500 THEN SetProgressPosition(cp, IDPROGRESSBAR, 500)
```

12.390SetProgressRange

Syntax

SetProgressRange(win as WINDOW,id as UINT,min as INT,max as INT)

Description

Sets the minimum and maximum values for a progress bar and redraws the bar to reflect the new range.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

min - A signed integer that represents the low limit to be set for the progress bar control.

max - A signed integer that represents the high limit to be set for the progress bar control.

Return value

None.

Remarks

The default progress bar range is 0 - 100.

Example usage

```
ProgressControl cp,40,80,236,20,@BORDER|@PBS_SMOOTH,0,IDPROGRESSBAR
SetProgressRange cp,IDPROGRESSBAR,0,500
SetProgressStep cp,IDPROGRESSBAR,50
```

12.391SetProgressStep

Syntax

SetProgressStep(win as WINDOW,id as UINT,value as INT)

Description

Specifies the step increment for a progress bar.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

value - The stepping increment.

Return value

None.

Remarks

The step increment is the amount by which the progress bar increases its current position whenever the [ProgressStepIt](#) command is called. The default is 10.

Example usage

```
ProgressControl cp,40,80,236,20,@BORDER|@PBS_SMOOTH,0,IDPROGRESSBAR
SetProgressRange cp,IDPROGRESSBAR,0,500
SetProgressStep cp,IDPROGRESSBAR,50
```

12.392SETSCROLLPOS

Syntax

SETSCROLLPOS(win as WINDOW,id as INT,pos as UINT)

Description

Sets the position of a scrollbar control or the windows scrollbars.

Parameters

win - Window or dialog containing the scrollbar to query.

id - Identifier of a control or windows scrollbar.

pos - New position for the scrollbar.

Return value

None

Remarks

Use and *id* of -1 for the windows horizontal scrollbar, -2 for the windows vertical scrollbar, or any other value for a scroll bar control. The new pos must be between the minimum and maximum range of the scrollbar.

See Also: [GETSCROLLPOS](#)

Example usage

```
SETSCROLLPOS win, -1, 50
```

12.393 SETSCROLLRANGE

Syntax

SETSCROLLRANGE(win as WINDOW, id as INT, min as INT, max as INT)

Description

Sets the scroll range of a scrollbar control or the windows scrollbars.

Parameters

win - Window or dialog containing the scrollbar control.

id - Identifier of the scrollbar control.

min - Minimum scroll position.

max - Maximum scroll position.

Return value

None.

Remarks

Use and *id* of -1 for the windows horizontal scrollbar, -2 for the windows vertical scrollbar, or any other value for a scroll bar control. All values returned by the scrollbar will be between *min* and *max*.

See Also: [GETSCROLLRANGE](#)

Example usage

```
SETSCROLLRANGE mywnd, 17, 0, 50
```

12.394 SETSELECTED

Syntax

SETSELECTED(win as WINDOW, id as INT, pos as INT)

Description

Sets the currently selected item in a list box or combo box control.

Parameters

win - Window or dialog containing the control.

id - Identifier of the control.

pos - Zero based string position to select.

Return value

None

Remarks

See Also: [GETSELECTED](#)

Example usage

```
SETSELECTED win, 17, 2
```

12.395SET SIZE

Syntax

SETSIZE(*win* as WINDOW, *l* as UINT, *t* as UINT, *w* as UINT, *h* as UINT, OPT *id*=0 as UINT)

Description

Sets the size of a window, dialog or control.

Parameters

win - Window or dialog.

l, *t*, *w*, *h* - New position and dimensions.

id - Optional control identifier.

Return value

None.

Remarks

See Also: [GETSIZE](#)

Example usage

```
SETSIZE mywindow, 0, 0, 100, 250
```

12.396SetSpinnerBase

Syntax

SetSpinnerBase(*win* as WINDOW, *id* as UINT, *base* as INT)

Description

Sets the current radix base for the spin button control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

base - New base value for the control. This argument can be 10 for decimal or 16 for hexadecimal.

Return value

None.

Remarks

The default is base 10.

Example usage

```
SetSpinnerBase demo, 300, 16
```

12.397 SetSpinnerBuddy

Syntax

```
SetSpinnerBuddy(win as WINDOW, id as UINT, buddy as UINT)
```

Description

Sets the current buddy control for the spin button control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

buddy - Identifier of the buddy control.

Return value

None.

Remarks

The buddy control is a control that is displayed along with the spin button control. For example, a common buddy control for a spin button is an edit control. You can obtain the identifier of the current buddy control by calling the [GetSpinnerBuddy](#) command. If the spin button does not have a buddy control associated with it, it acts as a simplified scroll bar.

Example usage

```
SetSpinnerBuddy demo, 800, IDEDIT1
```

12.398 SetSpinnerPosition

Syntax

```
SetSpinnerPosition(win as WINDOW, id as UINT, pos as INT)
```

Description

Sets the current position of the spin button control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

pos - Then new position.

Return value

None.

Remarks

The *pos* value must be within the range specified for the control with the [SetSpinnerRange](#) command.

Example usage

```
SetSpinnerPosition demo, 800, 0xFF
```

12.399SetSpinnerRange

Syntax

SetSpinnerRange(win as WINDOW,id as UINT,min as INT,max as INT)

Description

Sets the current range of the spin button control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

min - Specifies the lower limit of the spin button range.

max - Specifies the upper limit of the spin button range.

Return value

None.

Remarks

The range is inclusive of the min and max values.

Example usage

```
SetSpinnerRange demo, 300, 0, 0xFF
```

12.400SETSTATE

Syntax

SETSTATE(win as WINDOW,id as INT,state as INT)

Description

Sets or resets a checkbox or radio button control.

Parameters

win - Window or dialog containing the control.

id - Identifier of the control.

state - New state of the control.

Return value

None

Remarks

State can either be 0 to uncheck the control or 1 to check the control.

See Also: [GETSTATE](#)

Example usage

```
SETSTATE win, 1, 1
```

12.401 SetTrackBarLineSize

Syntax

SetTrackBarLineSize(win as WINDOW, id as UINT, size as INT)

Description

Sets the number of logical positions the TrackBar's slider moves in response to keyboard input from the arrow keys.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

size - New line size.

Return value

None.

Remarks

The logical positions are the integer increments in the trackbar's range of minimum to maximum slider positions. The default setting for the line size is 1.

Example usage

```
SetTrackbarLineSize cp, IDTRACKBAR, 2
```

12.402SetTrackBarPageSize

Syntax

SetTrackBarPageSize(win as WINDOW,id as UINT,size as INT)

Description

Sets the page size for a trackbar control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

size - New page size.

Return value

None.

Remarks

Sets the number of logical positions the trackbar's slider moves in response to keyboard input, such as the or keys, or mouse input, such as clicks in the trackbar's channel. The logical positions are the integer increments in the trackbar's range of minimum to maximum slider positions.

Example usage

```
SetTrackbarPageSize cp, IDTRACKBAR, 20
```

12.403SetTrackBarPosition

Syntax

SetTrackBarPosition(win as WINDOW,id as UINT,pos as INT)

Description

Sets the current logical position of the slider in a trackbar.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

pos - New position for the trackbar control.

Return value

None.

Remarks

Valid logical positions are the integer values in the trackbar's range of minimum to maximum slider positions. If this value is outside the control's maximum and minimum range, the position is set to

the maximum or minimum value.

Example usage

```
SetTrackBarPosition demo, 900, 50
```

12.404SetTrackBarRange

Syntax

SetTrackBarRange(win as WINDOW,id as UINT,min as INT,max as INT)

Description

Sets the range of minimum and maximum logical positions for the slider in a trackbar.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

min - Minimum position for the slider.

max - Maximum position for the slider.

Return value

None.

Remarks

If the current slider position is outside the new range, this method sets the slider position to the new maximum or minimum value.

Example usage

```
TrackbarControl cp,40,80,236,40,@TBS_AUTOTICKS|@TBS_FIXEDLENGTH|@TABSTOP|@BORDER,0,IDI_
SetTrackbarRange cp,IDTRACKBAR,-100,100
SetTrackbarTickFreq cp,IDTRACKBAR,10
SetTrackbarPageSize cp,IDTRACKBAR,20
SetTrackbarThumbLength cp,IDTRACKBAR,20
```

12.405SetTrackBarThumbLength

Syntax

SetTrackBarThumbLength(win as WINDOW,id as UINT,length as INT)

Description

Sets the length of the slider in a trackbar.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

length - New length of the thumb slider, in pixels.

Return value

None.

Remarks

This message is ignored if the trackbar does not have the `@TBS_FIXEDLENGTH` style.

Example usage

```
TrackbarControl cp,40,80,236,40,@TBS_AUTOTICKS|@TBS_FIXEDLENGTH|@TABSTOP|@BORDER,0,IDI
SetTrackbarRange cp,IDTRACKBAR,-100,100
SetTrackbarTickFreq cp,IDTRACKBAR,10
SetTrackbarPageSize cp,IDTRACKBAR,20
SetTrackbarThumbLength cp,IDTRACKBAR,20
```

12.406SetTrackBarTickFreq

Syntax

SetTrackBarTickFreq(win as WINDOW,id as UINT,freq as INT)

Description

Sets the interval frequency for tick marks in a trackbar.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

freq - Frequency of the tick marks.

Return value

None.

Remarks

For example, if the frequency is set to two, a tick mark is displayed for every other increment in the trackbar's range. The default setting for the frequency is one; that is, every increment in the range is associated with a tick mark. The trackbar must have the `@TBS_AUTOTICKS` style to use this command.

Example usage

```
TrackbarControl cp,40,80,236,40,@TBS_AUTOTICKS|@TBS_FIXEDLENGTH|@TABSTOP|@BORDER,0,IDI
SetTrackbarRange cp,IDTRACKBAR,-100,100
SetTrackbarTickFreq cp,IDTRACKBAR,10
SetTrackbarPageSize cp,IDTRACKBAR,20
SetTrackbarThumbLength cp,IDTRACKBAR,20
```

12.407SETTYPE

Syntax

SETTYPE ptr as POINTER, type

Description

Sets the type of a pointer for automatic typecasting.

Parameters

ptr - A pointer variable

type - A built in or UDT type name.

Return value

None

Remarks

Pointers in IW BASIC are generic and don't generally point to a specific type of data. When using NEW the pointers type will be set to any built in variable type however UDT's will need to use SETTYPE or typecasting. A pointer defined as a parameter in a subroutine always needs to be explicitly set to a type with SETTYPE or always use typecasting. Either method produces the same output code.

See Also: [TYPEOF](#)

Example usage

```
TYPE MYUDT
    DEF age as INT
    DEF name as STRING
ENDTYPE

DEF pUdt as POINTER
pUdt = NEW(MYUDT,1)

'without SETTYPE using typecasting
#<MYUDT>pUdt.name = "Joe Smith"
#<MYUDT>pUdt.age = 7

'with SETTYPE
SETTYPE pUdt, MYUDT
#pUdt.name = "Joe Smith"
#pUdt.age = 7
```

12.408 SETWINDOWCOLOR

Syntax

SETWINDOWCOLOR(win as WINDOW,colr as UINT)

Description

Changes the background color of a window.

Parameters

win - Window

colr - New background color

Return value

None

Remarks

None

Example usage

```
SETWINDOWCOLOR mywindow, RGB(0,0,0)
```

12.409SGN

Syntax

INT = SGN(num as DOUBLE)

Description

Determines the sign of a number.

Parameters

num - Number to test.

Return value

-1 if the number is less than zero, 1 if the number is greater than zero, or 0 if the number equals zero.

Remarks

None

Example usage

```
a = -1.223  
PRINT SGN(a)
```

12.410SHOWCONTEXTMENU

Syntax

SHOWCONTEXTMENU(win as WINDOW,hmenu as UINT,x as INT,y as INT)

Description

Displays a right-click context menu. Used internally by the menu creation macros.

Parameters

win - Window or dialog to show the menu.

hmenu - Handle to a menu returned by CREATEMENU.

x, y - Window coordinates of menu.

Return value

None

Remarks

See Also: [CREATEMENU](#)

Example usage

None

12.411 SHOWDIALOG

Syntax

SHOWDIALOG(*dlg* as DIALOG, *OPT parent* as DIALOG|WINDOW)

Description

Displays a non-modal dialog.

Parameters

dlg - Variable of type DIALOG initialized with CREATEDIALOG

parent - Optional. Specifies the parent window or dialog for this dialog. Overrides the parent parameter in the CREATEDIALOG statement

Return value

None

Remarks

Unlike DOMODAL a dialog shown with SHOWDIALOG does not block user input with other window. A non-modal dialog requires normal message processing with a WAIT or WAITUNTIL loop.

See Also: [CREATEDIALOG](#), [CLOSEDIALOG](#), [DOMODAL](#)

Example usage

```
SHOWDIALOG mydlg, mainwin
```

12.412 SHOWIMAGE

Syntax

SHOWIMAGE(*win* as WINDOW, *handle* as UINT, *type* as INT, *x* as INT, *y* as INT, *OPT w* as INT, *OPT h* as INT, *OPT flags* as INT)

Description

Draws a previously loaded IMAGE, ICON or EMF in the window specified.

Parameters

win - Window to display image into.

handle - Handle to image returned from LOADIMAGE.

type - Type used with LOADIMAGE.

x, y - Coordinate for upper left corner of image.

w, h - Optional width and height.

flags - Optional BitBlt flags for @IMGBITMAP type.

Return value

None

Remarks

The optional flags variable is for advanced use and are passed directly to the WIN32 BitBlt function. *w* and *h* allow scaling an image, if omitted the size of the image is determined from the file when loading. The @IMGICON type ignores *w* and *h*.

See Also: [LOADIMAGE](#)

Example usage

```
SHOWIMAGE win1, mybitmap, @IMGBITMAP, 0, 0
```

12.413SHOWWINDOW

Syntax

SHOWWINDOW(win as WINDOW,nCmdShow as INT,OPT id=0 as UINT)

Description

Shows or hides a window, dialog or control.

Parameters

win - Window or dialog.

nCmdShow - Show window command.

id - Optional control identifier.

Return value

None

Remarks

nCmdShow is one of:

@SWHIDE - Hides the window, dialog or control

@SWRESTORE - Restores the window, dialog or control

@SWMINIMIZED - Minimizes the dialog or window

@SWMAXIMIZED - Maximizes the dialog or window

@SWSHOW - Shows a hidden window

See Also: [SETSIZE](#)

Example usage

```
SHOWWINDOW mywnd, @SWHIDE
```

12.414 SIN

Syntax

DOUBLE = SIN(num as DOUBLE)

Description

Calculates the sine of an angle.

Parameters

num - The angle in radians to calculate the sine.

Return value

The sine of the angle *num*.

Remarks

See Also: [SIND](#), [FSIN](#), [FSIND](#)

Example usage

```
PRINT SIN(1)
```

12.415 SIND

Syntax

DOUBLE = SIND(num as DOUBLE)

Description

Calculates the sine of an angle.

Parameters

num - The angle in degrees to calculate the sine.

Return value

The sine of the angle *num*.

Remarks

See Also: [SIN](#), [FSIN](#), [FSIND](#)

Example usage

```
PRINT SIND(90)
```

12.416 SINH

Syntax

DOUBLE = SINH(num as DOUBLE)

Description

Calculates the hyperbolic sine of an angle.

Parameters

num - The angle in radians to calculate the hyperbolic sine.

Return value

The hyperbolic sine of the angle *num*.

Remarks

See Also: [FSINH](#), [SINH](#), [FSINH](#)

Example usage

```
PRINT SINH(1.223)
```

12.417SINH

Syntax

DOUBLE = SINHD(num as DOUBLE)

Description

Calculates the hyperbolic sine of an angle.

Parameters

num - The angle in degrees to calculate the hyperbolic sine.

Return value

The hyperbolic sine of the angle *num*.

Remarks

See Also: [SINH](#), [FSINH](#), [FSINH](#)

Example usage

```
PRINT SINHD(42.0)
```

12.418sizeof

Syntax

INT = sizeof(variable)

Description

Used to determine the size, in bytes, of memory reserved for a variable (numbers, arrays, strings, classes, ...)

Parameters

variable - The variable to be examined..

Return value

The size, in bytes, of memory reserved for the variable.

Remarks

SIZEOF will always evaluate at compile time, without generating code. SIZEOF(1) is 4, sizeof (string) is 255, and so on.

See Also: [LEN](#)

Example usage

```
iwstring iw[32]
' compute the maximum number of characters
const NumberOfCharacters = sizeof(iw) / sizeof(iw[0])
print NumberOfCharacters ' 32
```

12.419SPACE\$ / WSPACE\$

Syntax

STRING = SPACE\$(num as INT)
WSTRING = WSPACE\$(num as INT)

Description

Returns a string filled with spaces.

Parameters

num - Number of spaces to fill.

Return value

A string of spaces.

Remarks

See Also: [STRINGS](#)

Example usage

```
A$ = SPACE$(10)
```

12.420SpinnerControl

Syntax

UINT = SpinnerControl(win as WINDOW,l as INT,t as INT,w as INT,h as INT,flags as INT,exStyle as INT,id as UINT)

Description

Creates a spinner control. A spinner control control is a pair of arrow buttons that the user can click to increment or decrement a value, such as a scroll position or a number displayed in a companion control (called a buddy window).

Parameters

win - Parent window or dialog containing the control.

l, t, w, h - Position and dimension of the control.

flags - Styles for the control.

exStyle - Extended window styles for the control

id - Control identifier.

Return value

If used with a window the return value is the handle of the control. If used with a dialog the return value is 0 and the handle to the control can be obtained with GETCONTROLHANDLE during the @IDINITDIALOG message.

Remarks

The control can also be created using CONTROLEX with a window class of "msctls_updown32". The spin button control is also known as an *up/down* control.

Styles

@UDS_WRAP - Causes the position to "wrap" if it is incremented or decremented beyond the ending or beginning of the range.

@UDS_SETBUDDYINT - Automatically changes the text of the buddy.

@UDS_ALIGNRIGHT - Positions the up-down control next to the right edge of the buddy window. The width of the buddy window is decreased to accommodate the width of the up-down control.

@UDS_ALIGNLEFT - Positions the up-down control next to the left edge of the buddy window. The buddy window is moved to the right, and its width is decreased to accommodate the width of the up-down control.

@UDS_AUTOBUDDY - Automatically chooses the previous created control as its buddy.

@UDS_ARROWKEYS - Causes the up-down control to increment and decrement the position when the UP ARROW and DOWN ARROW keys are pressed.

@UDS_HORZ - Causes the up-down control's arrows to point left and right instead of up and down.

@UDS_NOTHOUSANDS - Does not insert a thousands separator between every three decimal digits.

@UDS_HOTTRACK - Causes the control to exhibit "hot tracking" behavior.

Example usage

```
SpinnerControl demo,0,0,14,34,@UDS_SETBUDDYINT | @UDS_AUTOBUDDY | @UDS_ALIGNRIGHT,0,30  
SetSpinnerRange demo, 300, 0, 0xFF
```

12.421SQRT

Syntax

DOUBLE = SQRT(num as DOUBLE)

Description

Returns the square root of a number.

Parameters

num - Number to take square root of.

Return value

Square root

Remarks

None

Example usage

```
sqr = SQR(100)
```

12.422STARTTIMER

Syntax

STARTTIMER(win as WINDOW,time as UINT,OPT id=1 as INT,OPT callback as UINT)

Description

Starts a timer in a window or dialog.

Parameters

win - Window or dialog

time - Timer interval in milliseconds

id - Optional timer id, defaults to 1.

callback - Optional timer callback address.

Return value

None

Remarks

The callback is an address of a subroutine to be called whenever the timer elapses. The subroutine must be declared as:

```
SUB name(hwnd as UINT, msg as UINT, idEvent as UINT, dwTime as UINT)
```

'name' can be anything you wish. hwnd will be the handle of the window that contains the timer, msg will always be @IDTIMER, idEvent is the timer id specified in the STARTTIMER statement and dwTime is the current system time in milliseconds.

If a callback address is not specified then the timer will send an @IDTIMER message to your window procedure at every interval. In this case @CODE (@WPARAM) will contain the timer id.

See Also: [STOPTIMER](#)

Example usage

```
'Start a timer with a callback
STARTTIMER win, 1000, 1, &mytimer
'Start a timer that sends @IDTIMER messages.
STARTTIMER win, 5000, 3
```

12.423STATIC

Syntax

```
STATIC DEF variable_name AS type
STATIC DEF variable_name : type
STATIC DIM variable_name AS type
STATIC DIM variable_name : type
STATIC type variable_name
```

Description

Declares a subroutine variable as being static.

Parameters

variable_name - The name of a variable to make static.

type - The type of a variable to make static.

Return value

None

Remarks

Static variables in a subroutine do not lose their contents, and act much like a global variable.

Example usage

```
sub function()
  POINT pt
  int x
  static def y as int
  static dim z as point
  '....
  return
endsub
```

12.424STDMETHOD

Syntax

```
STDMETHOD name(paramlist)
```

Description

Declares a COM method in an interface.

Parameters

paramlist - Comma separated list of parameters in the format of *variable as type* or *variable: type*.

Return value

None

Remarks

A method in a COM object is actually a function. The standard return type for all COM methods is an HRESULT which is a UINT value. The method declaration follows the same parameter list format as the DECLARE statement

See Also: [INTERFACE](#), [ENDINTERFACE](#), [SET INTERFACE](#), [DEFINE GUID](#), [DECLARE](#)

Example usage

```
INTERFACE IUnknown
    STDMETHOD QueryInterface(riid as POINTER, ppvObj as POINTER),INT
    STDMETHOD AddRef(),INT
    STDMETHOD Release(),INT
ENDINTERFACE
```

12.425STEP

Syntax

STEP

Description

Reserved word. Used in FOR/NEXT loops

Parameters

N/A

Return value

N/A

Remarks

See Also: [FOR](#), [NEXT](#)

Example usage

```
FOR x = 0 TO 100 STEP 2
NEXT x
```

12.426STOP

Syntax

STOP

Description

Inserts a breakpoint in your code for the debugger.

Parameters

None

Return value

None

Remarks

Ignored if not compiling under debug mode. When debugging the program will stop and return control to the built in debugger, display the line number and context information and enable the single step option.

Example usage

None

12.427STOPMIDI\$

Syntax

STOPMIDI\$(pThread as POINTER)

Description

Stops a currently playing stream of notes started with the PLAYMIDI\$ command.

Parameters

pThread - Pointer returned by PLAYMIDI\$.

Return value

None.

Remarks

For use with the asynchronous mode of PLAYMIDI\$. It is safe to call this command even if the stream has already finished playing.

See Also: [PLAYMIDI\\$](#), [Music and Sound commands](#)

Example usage

```
pThread = PLAYMIDI$("T120 N0 I25 O5 *C2EG;D4EFG*A2O6CE; *C2EG;D4EFG*A2O7CE;", TRUE)
DO:UNTIL INKEY$<> ""
STOPMIDI$(pThread)
```

12.428STOPTIMER

Syntax

STOPTIMER(win as WINDOW,OPT id=1 as INT)

Description

Stops a previously started timer.

Parameters

win - Window or dialog containing the timer.

id - Optional timer identifier, defaults to 1.

Return value

None

Remarks

The timer identifier must be the same one used to start the timer.

See Also: [STARTTIMER](#)

Example usage

```
STOPTIMER win, 3
```

12.429STR\$ /WSTR\$

Syntax

STRING = STR\$(num as DOUBLE)

WSTRING =WSTR\$(num as DOUBLE)

Description

Converts a number to a string.

Parameters

num - Number to convert.

Return value

String representation of number.

Remarks

SETPRECISION controls number of decimal places converted.

See Also: [USING](#), [SETPRECISION](#)

Example usage

```
A$ = STR$(1.23)
```

12.430STRING\$ /WSTRING\$

Syntax

STRING = STRING\$(count as INT,c as char)
WSTRING = WSTRING\$(count as INT,c as word)

Description

Returns a string filled with *count* number of the character specified.

Parameters

count - Number of characters to fill
c - Fill Character

Return value

String of duplicated characters

Remarks

See Also: [SPACES\\$](#)

Example usage

```
A$ = STRING$(5, "A")
```

12.431SUB

Syntax

SUB name(paramlist) { ,returntype }

Description

Begins a subroutine (function).

Parameters

name - Name of subroutine
paramlist - Comma separated list of parameters in the format of *variable as type* or *variable: type*.
returntype - Optional return type. Any built in type or UDT.

Return value

None

Remarks

Each SUB statement must be matched with an ENDSUB. You must use the RETURN statement whether or not you return a value. The return type is optional and can be omitted if you don't return a value.

See Also: [DECLARE](#), [ENDSUB](#), [RETURN](#)

Example usage

```
SUB mysub( a as INT, b as INT), INT
    RETURN a+b
ENDSUB
```

12.432SWORD**Syntax**

INT = SWORD(num)

Description

Converts a floating point value to an integer.

Parameters

num - An integer

Return value

An signed word

Remarks**Example usage**

Unpack two signed words packed into single integer (WM_MOUSEMOVE)

```
word w = word(integer)
```

12.433SYSTEM**Syntax**

SYSTEM(command as STRING, OPT param as STRING)

Description

Runs another executable.

Parameters

command - Executable to run.

param - Parameter for executable.

Return value

None

Remarks

Command must be full pathname to executable or the executable needs to be in the system path.

Example usage

```
SYSTEM "notepad.exe" , "c:\\mydocs\\prog.txt"
```

12.434TabControl

Syntax

UINT = TabControl(win as WINDOW,l as INT,t as INT,w as INT,h as INT,flags as INT,exStyle as INT,id as UINT)

Description

Creates a tab control.

Parameters

win - Parent window or dialog containing the control.

l, t, w, h - Position and dimension of the control.

flags - Styles for the control.

exStyle - Extended window styles for the control

id - Control identifier.

Return value

If used with a window the return value is the handle of the control. If used with a dialog the return value is 0 and the handle to the control can be obtained with GETCONTROLHANDLE during the @IDINITDIALOG message.

Remarks

The control can also be created using CONTROLEX with a window class of "systabcontrol32".

Styles

@TCS_SCROLLOPPOSITE - Unneeded tabs scroll to the opposite side of the control when a tab is selected.

@TCS_BOTTOM - Tabs appear at the bottom of the control.

@TCS_RIGHT - Tabs appear vertically on the right side of controls that use the

@TCS_VERTICAL style.

@TCS_MULTISELECT - Multiple tabs can be selected by holding down the CTRL key when clicking.

@TCS_FLATBUTTONS - Selected tabs appear as being indented into the background.

@TCS_FORCEICONLEFT - Icons are aligned with the left edge of each fixed-width tab. This style can only be used with the @TCS_FIXEDWIDTH style.

@TCS_FORCELABELLEFT - Labels are aligned with the left edge of each fixed-width tab. This style can only be used with the @TCS_FIXEDWIDTH style, and it implies the

@TCS_FORCEICONLEFT style.

@TCS_HOTTRACK - Items under the pointer are automatically highlighted.

@TCS_VERTICAL - Tabs appear at the left side of the control, with tab text displayed vertically.

@TCS_TABS - Tabs appear as tabs, and a border is drawn around the display area. This style is the default.

@TCS_BUTTONS - Tabs appear as buttons, and no border is drawn around the display area.

- @TCS_SINGLELINE - Only one row of tabs is displayed.
- @TCS_MULTILINE - Multiple rows of tabs are displayed, if necessary, so all tabs are visible at once.
- @TCS_RIGHTJUSTIFY - The width of each tab is increased, if necessary, so that each row of tabs fills the entire width of the tab control.
- @TCS_FIXEDWIDTH - All tabs are the same width. This style cannot be combined with the @TCS_RIGHTJUSTIFY style.
- @TCS_RAGGEDRIGHT - Rows of tabs will not be stretched to fill the entire width of the control. This style is the default.
- @TCS_FOCUSONBUTTONDOWN - The tab control receives the input focus when clicked.
- @TCS_OWNERDRAWFIXED - The parent window is responsible for drawing tabs.
- @TCS_TOOLTIPS - The tab control has a ToolTip control associated with it.
- @TCS_FOCUSNEVER - The tab control does not receive the input focus when clicked.

Example usage

```
TabControl cp,0,0,640,480,@BORDER|@TABSTOP|@TCS_TOOLTIPS|@TCS_HOTTRACK,0, IDTABCONTROL
tcInsertTab cp, IDTABCONTROL,0, "Calendar"
tcInsertTab cp, IDTABCONTROL,1, "Trackbar"
tcInsertTab cp, IDTABCONTROL,2, "Progress Bar"
tcInsertTab cp, IDTABCONTROL,3, "Date/Time picker"
tcInsertTab cp, IDTABCONTROL,4, "IP Address"
```

12.435TAN

Syntax

DOUBLE = TAN(num as DOUBLE)

Description

Returns the tangent of an angle.

Parameters

num - Angle in radians.

Return value

Tangent of angle *num*.

Remarks

See Also: [TAND](#), [FTAN](#), [FTAND](#)

Example usage

```
PRINT TAN(7.75)
```

12.436TAND

Syntax

DOUBLE = TAND(num as DOUBLE)

Description

Calculates the tangent of an angle.

Parameters

num - Angle in degrees.

Return value

Tangent of angle *num*.

Remarks

See Also: [TAN](#), [FTAN](#), [FTAND](#)

Example usage

```
PRINT TAND(43.8)
```

12.437TANH

Syntax

DOUBLE = TANH(num as DOUBLE)

Description

Calculates the hyperbolic tangent of an angle.

Parameters

num - Angle in radians.

Return value

Hyperbolic tangent of the angle *num*.

Remarks

See Also: [TANH](#), [FTANH](#), [FTANH](#)

Example usage

```
PRINT TANH(1.113)
```

12.438TANHD

Syntax

DOUBLE = TANHD(num as DOUBLE)

Description

Calculates the hyperbolic tangent of an angle.

Parameters

num - Angle in degrees.

Return value

The hyperbolic tangent of the angle *num*.

Remarks

See Also: [TANH](#), [FTANH](#), [FTANHHD](#)

Example usage

```
PRINT TANHD(47.56)
```

12.439tcDeleteAllTabs

Syntax

tcDeleteAllTabs(win as WINDOW,id as UINT)

Description

Removes all tabs from a tab control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

None.

Remarks

None.

Example usage

```
tcDeleteAllTabs demo, IDTABCONTROL
```

12.440tcDeleteTab

Syntax

tcDeleteTab(win as WINDOW,id as UINT,item as INT)

Description

Removes a tab from a tab control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

item - Zero based index of tab.

Return value

None.

Remarks

None.

Example usage

```
tcDeleteTab demo, IDTABCONTROL, 1
```

12.441tcGetFocusTab

Syntax

```
INT = tcGetFocusTab(win as WINDOW,id as UINT)
```

Description

Call this function to retrieve the index of the tab with the current focus.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

The zero-based index of the tab with the current focus.

Remarks

None.

Example usage

```
current = tcGetFocusTab(demo, IDTABCONTROL)
```

12.442tcGetItemData

Syntax

```
UINT = tcGetItemData(win as WINDOW,id as UINT,tab as INT)
```

Description

Retrieves the application defined data associated with a tab.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

tab - Zero based index of tab.

Return value

Application-defined data associated with the tab.

Remarks

An applications sets the item data with the [tcSetItemData](#) command.

Example usage

```
handle = tcGetItemData(demo, IDTABCONTROL, 1)
```

12.443tcGetRowCount

Syntax

INT = tcGetRowCount(win as WINDOW,id as UINT)

Description

Retrieves the current number of rows in a tab control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

The current number of rows.

Remarks

Only tab controls that have the @TCS_MULTILINE style can have multiple rows of tabs.

Example usage

```
count = tcGetRowCount(demo, IDTABCONTROL)
```

12.444tcGetSelectedTab

Syntax

INT = tcGetSelectedTab(win as WINDOW,id as UINT)

Description

Retrieves the index of the currently selected tab.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

Zero-based index of the selected tab if successful or -1 if no tab is selected.

Remarks

None.

Example usage

```
sel = tcGetSelectedTab(demo, IDTABCONTROL)
```

12.445tcGetTabCount

Syntax

```
INT = tcGetTabCount(win as WINDOW,id as UINT)
```

Description

Retrieves the number of tabs in the control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

Return value

Number of items in the tab control.

Remarks

None.

Example usage

```
count = tcGetTabCount(demo, IDTABCONTROL)
```

12.446tcGetTabText

Syntax

```
STRING = tcGetTabText(win as WINDOW,id as UINT,item as INT)
```

Description

Returns the text of a tab.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

item - Zero based index of tab.

Return value

String containing the tabs text.

Remarks

None.

Example usage

```
text = tcGetTabText(demo, IDTABCONTROL)
```

12.447tcHighlightTab

Syntax

tcHighlightTab(win as WINDOW,id as UINT,item as INT,bHighlight as INT)

Description

Sets the highlight state of a tab item.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

item - Zero based index of tab.

bHighlight - Value specifying the highlight state to be set.

Return value

None.

Remarks

If *bHighlight* is TRUE, the tab is highlighted; if FALSE, the tab is set to its default state.

Example usage

```
tcHighlightTab demo, IDTABCONTROL, 0, TRUE
```

12.448tcHitTest

Syntax

INT = tcHitTest(win as WINDOW,id as UINT,x as INT,y as INT)

Description

Determines which tab, if any, is at a specified screen position.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

x - X position to test.

y - Y position to test.

Return value

Returns the index of the tab, or -1 if no tab is at the specified position.

Remarks

X and Y are specified in screen coordinates.

Example usage

```
tab = tcHitTest(demo, IDTABCONTROL, 100,100)
```

12.449tcInsertTab

Syntax

tcInsertTab(win as WINDOW,id as UINT,index as INT,item as STRING,OPT iImage = -1 as INT)

Description

Inserts a new tab in a tab control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

index - Index of the new tab.

item - Text of the new tab.

iImage - Optional. Index in the tab control's image list, or -1 if there is no image for the tab.

Return value

None.

Remarks

None.

Example usage

```
TabControl cp,0,0,640,480,@BORDER|@TABSTOP|@TCS_TOOLTIPS|@TCS_HOTTRACK,0, IDTABCONTROL
tcInsertTab cp, IDTABCONTROL,0, "Calendar"
tcInsertTab cp, IDTABCONTROL,1, "Trackbar"
tcInsertTab cp, IDTABCONTROL,2, "Progress Bar"
tcInsertTab cp, IDTABCONTROL,3, "Date/Time picker"
tcInsertTab cp, IDTABCONTROL,4, "IP Address"
```

12.450tcSetFocusTab

Syntax

tcSetFocusTab(win as WINDOW,id as UINT,item as INT)

Description

Sets the focus to a specified tab in a tab control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

item - Zero based index of tab.

Return value

None.

Remarks

If the tab control has the `@TCS_BUTTONS` style (button mode), the tab with the focus may be different from the selected tab. For example, when a tab is selected, the user can press the arrow keys to set the focus to a different tab without changing the selected tab.

If the tab control does not have the `@TCS_BUTTONS` style, changing the focus also changes the selected tab.

Example usage

```
tcSetFocusTab demo, IDTABCONTROL, 3
```

12.451tcSetImage

Syntax

```
tcSetImage(win as WINDOW,id as UINT,tab as INT,image as INT)
```

Description

Changes or removes a tabs image.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

tab - Zero based index of tab.

image - Index in the tab control's image list, or -1 if there is no image for the tab.

Return value

None.

Remarks

The image list must have been set in the control using the [tcSetImageList](#) command.

Example usage

```
tcSetImage demo, IDTABCONTROL, 1, 1
```

12.452tcSetImageList

Syntax

```
tcSetImageList(win as WINDOW,id as UINT,himl as UINT)
```

Description

Assigns an image list to a tab control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

himl - Handle to an image list created with the `ImageList_Create` API function.

Return value

None.

Remarks

None.

Example usage

```
tcSetImageList demo, IDTABCONTROL, hIml
```

12.453tcSetItemData

Syntax

tcSetItemData(win as WINDOW,id as UINT,tab as INT,value as UINT)

Description

Sets the application-defined item data associated with a tab in the tab control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

tab - Zero based index of tab.

value - User-defined data value.

Return value

None.

Remarks

The value can be retrieved using the [tcGetItemData](#) command.

Example usage

```
tcSetItemData demo, IDTABCONTROL, 1, handle
```

12.454tcSetMinTabSize

Syntax

tcSetMinTabSize(win as WINDOW,id as UINT,cx as INT)

Description

Sets the minimum width of items in a tab control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

cx - Minimum width to be set for a tab control item. If this parameter is set to -1, the control will use the default tab width.

Return value

None.

Remarks

Width is in pixels.

Example usage

```
tcSetMinTabSize demo, IDTABCONTROL, 60
```

12.455 tcSetSelectedTab

Syntax

```
tcSetSelectedTab(win as WINDOW, id as UINT, item as INT)
```

Description

Selects a tab in a tab control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

item - Zero based index of tab.

Return value

None.

Remarks

A tab control does not send a @TCN_SELCHANGING or @TCN_SELCHANGE notification message when a tab is selected using this message.

Example usage

```
tcSetSelectedTab demo, IDTABCONTROL, 1
```

12.456 tcSetTabText

Syntax

```
tcSetTabText(win as WINDOW, id as UINT, item as INT, text as STRING)
```

Description

Changes the text of a tab.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

item - Zero based index of tab.

text - New tab text.

Return value

None.

Remarks

None.

Example usage

```
tcSetTabText demo, IDTABCONTROL, 4, "Preview"
```

12.457tcSetTip

tcSetTip

tcSetTip(win as WINDOW,id as UINT,tab as INT,text as STRING)

Description

Sets the tool tip for a tab in the tab control.

Parameters

win - Dialog or window containing the control.

id - Identifier of the control.

tab - Zero based index of tab.

text - New tool tip text for the tab.

Return value

None.

Remarks

The tab control must have been created with the @TCS_TOOLTIPS style.

Example usage

```
tcSetTip cp, IDTABCONTROL, 0, tcGetTabText (cp, IDTABCONTROL, 0)
tcSetTip cp, IDTABCONTROL, 1, tcGetTabText (cp, IDTABCONTROL, 1)
tcSetTip cp, IDTABCONTROL, 2, tcGetTabText (cp, IDTABCONTROL, 2)
```

12.458THEN

Syntax

THEN

Description

Reserved word. Used in IF statements and IF blocks.

Parameters

None

Return value

None

Remarks

See Also: [IF](#)

Example usage

```
IF a = 3 THEN b = 7
```

12.459THREAD

Syntax

THREAD *typevariable* *variablename* {,*variablename*}

Description

Used to create thread-private variables in multi-thread processes..

Parameters

typevariable - The type of variable being defined (INT, STRING, etc.).

variablename - The variables being defined.

Return value

None

Remarks

None

Example usage

See [Language/Threads](#)

12.460THROW

Syntax

THROW *value*

Description

Used to create a user generated exception.

Parameters

value - Can be a variable, a literal number, or a literal string.

Return value

None

Remarks

See Also: [TRY](#), [ENDTRY](#), [CATCH](#), [ENDCATCH](#), [Exception Handling](#)

Example usage

See [Exception Handling](#)

12.461TIME\$

Syntax

STRING = TIME\$

Description

Returns the current time.

Parameters

None.

Return value

String containing the current time in the format HH:MM:SS

Remarks

See Also: [DATES](#)

Example usage

```
PRINT TIME$
```

12.462TIMER

Syntax

INT = TIMER

Description

Returns the number of seconds elapsed since Windows was started.

Parameters

None.

Return value

Number of seconds.

Remarks

None.

Example usage

```
i = TIMER.
```

12.463TO

Syntax

TO

Description

Reserved word. Used in FOR/NEXT loops.

Parameters

N/A

Return value

N/A

Remarks

See Also: [FOR](#), [NEXT](#)

Example usage

```
FOR x = 1 TO 10  
NEXT x
```

12.464ToolTipControl

Syntax

UINT = ToolTipControl(win as WINDOW, flags as INT, exStyle as INT)

Description

Creates a tool tip control.

Parameters

win - Parent window or dialog containing the control.

flags - Styles for the control.

exStyle - Extended window styles for the control.

Return value

Handle to the tool tip control.

Remarks

Tooltip windows aren't created like most other windows, and have specific styles that should be set. For dialogs create the tool tip control in response to the @IDINTIDIALOG message. A window/dialog only needs one tool tip control to display tips for all child controls.

Style

@TTS_ALWAYSSTIP - Indicates that the ToolTip control appears when the cursor is on a tool, even if the ToolTip control's owner window is inactive.

@TTS_NOPREFIX - Prevents the system from stripping ampersand characters from a string or terminating a string at a tab character.

@TTS_NOFADE - Disables fading ToolTip animation on Windows 2000 and above systems.

@TTS_NOANIMATE - Disables sliding ToolTip animation on Microsoft Windows 2000 and above systems.

@TTS_BALLOON - Indicates that the ToolTip control has the appearance of a cartoon "balloon".

Example usage

```
'create the tool tip control
hTooltip = ToolTipControl(cp,@TTS_ALWAYSSTIP|@TTS_BALLOON,0)
'add tooltips for our buttons
ttAddTool hTooltip,@TTF_SUBCLASS|@TTF_IDISHWND|@TTF_CENTERTIP,GetControlHandle(cp,1),c
ttAddTool hTooltip,@TTF_SUBCLASS|@TTF_IDISHWND|@TTF_CENTERTIP,GetControlHandle(cp,2),c
```

12.465TrackBarControl**Syntax**

UINT = TrackBarControl(win as WINDOW,l as INT,t as INT,w as INT,h as INT,flags as INT,exStyle as INT,id as UINT)

Description

Creates a Track Bar control.

Parameters

win - Parent window or dialog containing the control.

l, t, w, h - Position and dimension of the control.

flags - Styles for the control.

exStyle - Extended window styles for the control

id - Control identifier.

Return value

If used with a window the return value is the handle of the control. If used with a dialog the return value is 0 and the handle to the control can be obtained with GETCONTROLHANDLE during the @IDINITDIALOG message.

Remarks

The control can also be created using CONTROLEX with a window class of "msctls_trackbar32".

Styles

@TBS_AUTOTICKS - The trackbar control has a tick mark for each increment in its range of

values.

@TBS_VERT - The trackbar control is oriented vertically.

@TBS_HORZ - The trackbar control is oriented horizontally. This is the default orientation.

@TBS_TOP - The trackbar control displays tick marks above the control. This style is valid only with @TBS_HORZ.

@TBS_BOTTOM - The trackbar control displays tick marks below the control. This style is valid only with @TBS_HORZ.

@TBS_LEFT - The trackbar control displays tick marks to the left of the control. This style is valid only with @TBS_VERT.

@TBS_RIGHT - The trackbar control displays tick marks to the right of the control. This style is valid only with @TBS_VERT.

@TBS_BOTH - The trackbar control displays tick marks on both sides of the control. This will be both top and bottom when used with @TBS_HORZ or both left and right if used with @TBS_VERT.

@TBS_NOTICKS - The trackbar control does not display any tick marks.

@TBS_ENABLESELRANGE - The trackbar control displays a selection range only.

@TBS_FIXEDLENGTH - The trackbar control allows the size of the slider to be changed with the [SetTrackBarThumbLength](#) command.

@TBS_NOHUMB - The trackbar control does not display a slider.

@TBS_TOOLTIPS - The trackbar control supports ToolTips. When a trackbar control is created using this style, it automatically creates a default Tooltip control that displays the slider's current position.

Example usage

```
TrackbarControl cp,40,80,236,40,@TBS_AUTOTICKS|@TBS_FIXEDLENGTH|@TABSTOP|@BORDER,0,IDI
SetTrackbarRange cp,IDTRACKBAR,-100,100
SetTrackbarTickFreq cp,IDTRACKBAR,10
SetTrackbarPageSize cp,IDTRACKBAR,20
SetTrackbarThumbLength cp,IDTRACKBAR,20
```

12.466TRY

Syntax

TRY

Description

The start of a block of code that traps exceptions..

Parameters

None

Return value

None

Remarks

See Also: [ENDTRY](#), [Exception Handling](#)

Example usage

```
exec_a:
  try
    print 1/a
  endtry
  catch
    a=1
    goto exec_a
  endcatch
```

12.467ttSetToolRect

Syntax

ttSetToolRect(*hToolTip* as UINT,*uid* AS UINT,*hwnd* as UINT,*rc* as WINRECT)

Description

Sets a new bounding rectangle for a tool.

Parameters

hToolTip - Handle to the tool tip control.

uid - Application-defined identifier of the tool.

hwnd - Parent window handle of the tool tip control.

rc - New bounding rectangle.

Return value

None.

Remarks

Used for defining a rectangular area in the parent window for a tool tip to show.

Example usage

```
ttAddTool hToolTip,0,100,cp.hwnd,"Black"
rcTool.left = 20:rcTool.top = 100:rcTool.right = rcTool.left+50:rcTool.bottom = rcTool.top+50
ttSetToolRect hToolTip,100,cp.hwnd,rcTool
```

12.468ttRelayMessage

Syntax

ttRelayMessage(*hToolTip* as UINT)

Description

Passes a mouse message to a ToolTip control for processing.

Parameters

hToolTip - Handle to the tool tip control.

Return value

None.

Remarks

When using rectangular areas for tool tips you must call this function when receiving the following messages in your parent window handler:

```
@IDMOUSEMOVE  
@IDLBUTTONDN  
@IDLBUTTONUP  
@IDRBUTTONDN  
@IDRBUTTONUP
```

Example usage

```
...  
CASE @IDMOUSEMOVE  
CASE& @IDLBUTTONDN  
CASE& @IDLBUTTONUP  
CASE& @IDRBUTTONDN  
CASE& @IDRBUTTONUP  
    'forward messages to the tooltip control  
    'so rectangular areas work  
    ttRelayMessage hToolTip
```

12.469ttDeleteTool

Syntax

```
ttDeleteTool(hToolTip as UINT,uID as UINT,hwnd as UINT)
```

Description

Removes a tip from a tool tip control.

Parameters

hToolTip - Handle to the tool tip control.

uID - Application-defined identifier of the tool.

hwnd - Parent window handle of the tool tip control.

Return value

None.

Remarks

None.

Example usage

```
ttDeleteTool hToolTip, 100, cp.hwnd
```

12.470ttAddTool**Syntax**

ttAddTool(hToolTip as UINT, flags as INT, uID as UINT, hwnd as UINT, text as POINTER)

Description

Adds a tip to a tool tip control.

Parameters

hToolTip - Handle to the tool tip control.

flags - Flags that control the ToolTip display.

uID - If *flags* includes the @TTF_IDISHWND flag, *uID* must specify the window handle to the tool.

hwnd - Handle to parent window.

text - The text for the tip.

Return value

None.

Remarks

Flags can be a combination of:

@TTF_IDISHWND - Indicates that the *uID* member is the window handle to the tool. If this flag is not set, *uID* is the tool's identifier.

@TTF_CENTERTIP - Centers the ToolTip window below the tool specified by the *uID* member.

@TTF_RTLREADING - Indicates that the ToolTip text will be displayed in the opposite direction to the text in the parent window.

@TTF_SUBCLASS - Indicates that the ToolTip control should subclass the tool's window to intercept messages.

@TTF_TRACK - Positions the ToolTip window next to the tool to which it corresponds.

@TTF_TRANSPARENT - Causes the ToolTip control to forward mouse event messages to the parent window. This is limited to mouse events that occur within the bounds of the ToolTip window.

Example usage

```
'create the tool tip control
hToolTip = ToolTipControl(cp, @TTS_ALWAYSSTIP|@TTS_BALLOON, 0)
'add tooltips for our buttons
ttAddTool hToolTip, @TTF_SUBCLASS|@TTF_IDISHWND|@TTF_CENTERTIP, GetControlHandle(cp, 1), c
ttAddTool hToolTip, @TTF_SUBCLASS|@TTF_IDISHWND|@TTF_CENTERTIP, GetControlHandle(cp, 2), c
```

12.471 tvDeleteAllItems

Syntax

INT = tvDeleteAllItems(win as WINDOW, id as UINT)

Description

Removes all items from a tree view control.

Parameters

win - Window or dialog containing control.

id - Identifier of tree view control.

Return value

Returns TRUE (1) if successful, FALSE(0) on failure.

Remarks

See Also: [tvDeleteItem](#)

Example usage

```
tvDeleteAllItems mywnd, 5
```

12.472 tvDeleteItem

Syntax

INT = tvDeleteItem(win as WINDOW, id as UINT, handle as UINT)

Description

Deletes an item from a tree view control.

Parameters

win - Window or dialog containing the control.

id - Identifier of the tree view control.

handle - Handle returned by tvInsertItem

Return value

Returns TRUE (1) if successful, FALSE(0) on failure.

Remarks

See Also: [tvDeleteAllItems](#), [tvInsertItem](#).

Example usage

```
tvDeleteItem mydlg, 17, hChild
```

12.473 tvGetItemData

Syntax

UINT = tvGetItemData(win as WINDOW,id as UINT,handle as UINT)

Description

Retrieves the 32 bit user data value associated with the tree view item.

Parameters

win - Window or dialog containing the control.

id - Identifier of the treeview control.

handle - Handle of the item.

Return value

Returns the 32 bit data value.

Remarks

See Also: [tvSetItemData](#).

Example usage

```
mydata = tvGetItemData(mydlg, 17, hChild)
```

12.474 tvGetItemText

Syntax

UINT = tvGetItemText(win as WINDOW,id as UINT,handle as UINT,strText as STRING,
cchTextMax as UINT)

Description

Retrieves the text of an item in a tree view control.

Parameters

win - [in] Window or Dialog containing the control.

id - [in] Identifier of tree view control.

handle - [in] Handle to item.

strText - [out] String to store item text into.

cchTextMax -[in] Maximum size of strText.

Return value

Returns TRUE (1) if successful, FALSE(0) on failure.

Remarks

See Also: [tvSetItemText](#)

Example usage

```
tvGetItemText(mydlg, 17, hChild, A$, 255)
```

12.475 tvGetSelectedItem

Syntax

UINT = tvGetSelectedItem(win as WINDOW, id as UINT)

Description

Returns a handle to the selected item in a tree view control.

Parameters

win - Window or dialog containing the control.

id - Identifier of the tree view control.

Return value

Handle to selected item or NULL if no item is currently selected.

Remarks

See Also: [tvSelectItem](#)

Example usage

```
hItem = tvGetSelectedItem(mydlg, 17)
```

12.476 tvInsertItem

Syntax

UINT = tvInsertItem(win as WINDOW, id as INT, strText as STRING, parent as UINT)

Description

Inserts a new item into the tree view control.

Parameters

win - Window or dialog containing the control.

id - Identifier of the tree view control.

strText - Text of the new item.

parent - Handle to the parent item or NULL to insert a new root item.

Return value

Returns TRUE (1) if successful, FALSE (0) on failure.

Remarks

See Also: [tvDeleteItem](#)

Example usage

```
root1 = tvInsertItem(win, 11, "Item 1", 0)
child1 = tvInsertItem(win, 11, "child 1", root1)
tvInsertItem(win, 11, "Child of child1", child1)
```

12.477 tvSelectItem

Syntax

INT = tvSelectItem(win as WINDOW,id as UINT,handle as UINT)

Description

Changes the selected item in a tree view control.

Parameters

win - Window or dialog containing the control.

id - Identifier of the tree view control.

handle - Handle to new selected item.

Return value

TRUE (1) if successful, FALSE(0) on failure.

Remarks

See Also: [tvGetSelectedItem](#)

Example usage

```
root1 = tvInsertItem(win,11,"Item 1",0)
child1 = tvInsertItem(win,11,"child 1",root1)
tvInsertItem(win,11,"Child of child1",child1)
tvSelectItem win, 11, root1
```

12.478 tvSetItemData

Syntax

UINT = tvSetItemData(win as WINDOW,id as UINT,handle as UINT,nData as UINT)

Description

Associates a 32 bit data value with a tree view item.

Parameters

win - Window or dialog containing the control.

id - Identifier of the tree view control.

handle - Handle of item.

nData - New 32 bit data value to associate with the item.

Return value

TRUE (1) if successful, FALSE (0) on failure.

Remarks

The 32 bit data value can be any integer value you wish to associate with a tree view item.

See Also: [tvGetItemData](#)

Example usage

```
tvSetItemData(mydlg, 17, hItem, 9999)
```

12.479 tvSetItemText**Syntax**

UINT = tvSetItemText(win as WINDOW, id as UINT, handle as UINT, strText as STRING)

Description

Changes the text of an item in a tree view control.

Parameters

win - Window or dialog containing the control.

id - Identifier of the tree view control.

handle - Handle to item.

strText - New item text.

Return value

TRUE (1) if successful, FALSE(0) on failure.

Remarks

See Also: [tvGetItemText](#)

Example usage

```
root1 = tvInsertItem(win, 11, "Item 1", 0)
child1 = tvInsertItem(win, 11, "child 1", root1)
tvSetItemText(win, 11, child1, "Files")
```

12.480 TYPE**Syntax**

TYPE name, OPT pack = 8

Description

Begins definition of a user defined data type (UDT)

Parameters

name - Name of the user defined type

pack - Optional byte packing value. Defaults to 8.

Return value

None

Remarks

The members of a UDT can be any built-in or user defined type and use the DEF statement for

defining. This includes multi-level nesting of TYPE/ENDTYPE definitions. Members can also be unions. You must end the definition with the ENDTYPE statement.

See Also: [ENDTYPE](#) , [UNION](#)

Example usage

```
TYPE foo
  DEF name as STRING
  DEF age as INT
ENDTYPE

DEF record as foo
record.name = "John Smith"
```

12.481 TYPEOF

Syntax

type = TYPEOF(variable)

Description

Returns the type of a variable.

Parameters

variable - Any defined variable.

Return value

The type of the variable.

Remarks

Generally used for the ANYTYPE subroutine parameter. Pointers will return a valid type only when used locally or specifically set with SETTYPE.

Returns one of the following constants:

- @TYPECHAR
- @TYPEINT
- @TYPEFLOAT
- @TYPEMEMORY
- @TYPEFILE
- @TYPEBFILE
- @TYPEWORD
- @TYPEDOUBLE
- @TYPEUSER
- @TYPEPOINTER
- @TYPEINT64
- @TYPEUINT64
- @TYPEUINT

See Also: [SETTYPE](#), [ISREF](#)

Example usage

```
SUB mysub(v as ANYTYPE)
DEF param as INT
  SELECT TYPEOF(v)
    CASE @TYPEINT
    CASE& @TYPEUINT
    IF ISREF(v) THEN param = ##<INT>v ELSE param = #<INT>v
...

```

12.482 UCASE\$ / WUCASE\$

Syntax

STRING = UCASE\$(str as STRING)
WSTRING = WUCASE\$(str as WSTRING)

Description

Converts a string to all upper case.

Parameters

str - The string to convert.

Return value

A copy of the input string converted to upper case.

Remarks

The input string remains unchanged.

See Also: [LCASE\\$](#)

Example usage

```
PRINT UCASE$("convert to upper case please")
```

12.483 UINT

Syntax

UINT = UINT(num)

Description

Converts a value to an unsigned integer.

Parameters

num - A number

Return value

An unsigned integer

Remarks

Example usage

```
PRINT UINT(1.56)
```

12.484UINT64

Syntax

UINT64 = UINT64(num)

Description

Converts a value to an unsigned UINT64.

Parameters

num - A number

Return value

An UINT64 value

Remarks

Example usage

```
PRINT UINT64(1.56)
```

12.485UNION

Syntax

UNION name

Description

Begins definition of a data union.

Parameters

name - Name of the union. Optional when union is part of a TYPE definition.

Return value

None

Remarks

The members of a UDT can be any built-in or user defined type and use the DEF statement for defining. You must end the definition with the ENDTYPE statement.

See Also: [ENDUNION](#)

Example usage

```
UNION k
  string name
  int id
ENDUNION

DEF y1,y2 AS k
y1.k.name = "Smith"
y2.k.id = 12345
```

12.486UNTIL

Syntax

UNTIL condition

Description

Terminates a DO/UNTIL loop and checks the condition.

Parameters

condition - Any condition that will be either TRUE or FALSE.

Return value

None

Remarks

If the *condition* is true UNTIL will continue executing at the first statement following the DO command.

To break out of a DO/UNTIL loop early you can use the [BREAK](#) command.

See Also: [DO](#)

Example usage

```
DO
  a = a + 1
  if a = 12 then BREAK
UNTIL a = 50
```

12.487USING / WUSING

Syntax

```
STRING = USING(format as STRING,...)
WSTRING = USING(format as WSTRING,...)
```

Description

Formats data.

Parameters

format - Input specifier string.
... - Variable list of parameters.

Return value

A string with the formatted data.

Remarks

See Also: [STR\\$](#)

Example usage

```
PRINT USING("#####.##  #####.##", total, subtotal)
```

12.488VAL

Syntax

DOUBLE = VAL(str as STRING)
DOUBLE = WVAL(str as WSTRING)

Description

Returns the numeric value of a string representation of a number.

Parameters

str - String to convert

Return value

Numeric value of string

Remarks

See Also: [STR\\$](#)

Example usage

```
num = VAL("123.44")
```

12.489W2S

Syntax

STRING = W2S(in as WSTRING)

Description

Converts a wide character (Unicode) string to an ANSI string. .

Parameters

in - Unicode string to convert..

Return value

The converted string..

Remarks

None

See Also: [S2W](#)

Example usage

```
PRINT W2S(L"A string to convert")
```

12.490WAIT

Syntax

WAIT(OPT nosleep=0 as INT)

Description

Waits for a window event.

Parameters

nosleep - Optional sleep parameter.

Return value

None

Remarks

Using *nosleep* = 1 checks for messages and returns immediately if none are available. WAIT will normally check for messages and then put the process in a sleep state until a message is available.

WAIT should only be used in special circumstances where you need more control of when messages are processed.

See Also: [WAITUNTIL](#)

Example usage

```
DO  
WAIT  
UNTIL done = 1
```

12.491WAITCON

Syntax

WAITCON

Description

Waits for a key to be pressed in the console

Parameters

None

Return value

None

Remarks

WAITCON pauses the console in a system friendly way. The input buffer is not cleared after this command has returned so it is effective to use INKEY\$ directly afterwards to see what key was read.

See Also: INKEY\$

Example usage

```
PRINT "Press any key to close"  
WAITCON
```

12.492WAITUNTIL

Syntax

WAITUNTIL condition

Description

Creates the main message loop of a Windows program, continually processes messages for all of the windows, dialogs and controls in your program until the condition is true. When a message arrives for one of your windows the procedure subroutine specified in the OPENWINDOW or CREATEDIALOG statements is called.

Parameters

condition - Any conditional statement that evaluates to TRUE (1) or FALSE (0)

Return value

None

Remarks

WAITUNTIL is functionally equivalent to:

```
DO  
WAIT  
UNTIL condition
```

See Also: [OPENWINDOW](#), [CREATEDIALOG](#), [WAIT](#)

Example usage

```
...  
exit = FALSE  
WAITUNTIL exit = TRUE
```

12.493WEND

Syntax

WEND

Description

Reserved word. Synonym for the ENDWHILE statement.

Parameters

None

Return value

None

Remarks

See [ENDWHILE](#)

12.494WHILE

Syntax

WHILE condition

Description

Begins a WHILE loop. As long as the condition is true, all of the statements between WHILE and ENDWHILE / WEND are executed in a loop.

Parameters

condition - Any condition that will be either TRUE or FALSE. (s)char and (s)word can be passed as a condition.

Return value

None

Remarks

To break out of a WHILE/ENDWHILE loop early you can use the [BREAK](#) command.

See Also: [ENDWHILE](#), [WEND](#), [BREAK](#)

Example usage

```
WHILE a < 7
  a = a + 1
```

```
ENDWHILE  
  
char c = 2  
  
while (c)  
    print c  
endwhile
```

12.495WORD

Syntax

INT = WORD(num)

Description

Converts a floating point value to an integer.

Parameters

num - An integer

Return value

An unsigned word value

Remarks

Example usage

Unpack two unsigned words packed into single integer

```
int x = word(lParam)  
int y = word(lParam>>16)
```

12.496WRITE

Syntax

INT = WRITE(vFile as FILE | BFILE, var as ANYTYPE)

Description

Writes data to a file.

Parameters

vFile - FILE or BFILE variable successfully opened with OPENFILE

var - For ASCII files can be any built in type. For binary files (BFILE) can be any built in type or UDT.

Return value

Returns number of bytes written to file.

Remarks

In ASCII mode WRITE separates numeric data with a space and strings are terminated with new line characters. In binary mode (BFILE) data is written in raw form.

See Also: [READ](#), [OPENFILE](#), [GET](#), [PUT](#)

Example usage

```
WRITE myfile, iNumber
```

12.497WRITEMEM

Syntax

WRITEMEM(*mptr* as MEMORY,*record* as INT,*var* as POINTER)

Description

Writes one record to allocated memory.

Parameters

mptr - A MEMORY variable successfully initialized with ALLOCMEM or returned by an API function.

record - The ones based record number.

var - Variable to save into memory. Any built in or user defined type (UDT).

Return value

None

Remarks

See Also: [READMEM](#), [ALLOCMEM](#), [FREEMEM](#)

Example usage

```
DEF buffer as MEMORY
DEF num as INT
ALLOCMEM buffer, 100, LEN(num)
FOR x = 1 to 100
    WRITEMEM buffer, x, x
NEXT x

FOR x = 1 to 100
    READMEM buffer, x, num
    PRINT num
NEXT x

FREEMEM buffer
```

12.498WRITEPRINTER

Syntax

WRITEPRINTER(*handle* as UINT,*data* as ANYTYPE)

Description

Writes data to printer.

Parameters

handle - Handle to printer returned by OPENPRINTER

data - Any built in data type.

Return value

None

Remarks

Numeric data types are converted to text with STR\$ before printing, SETPRECISION controls number of decimal places printed. Variable types such as POINTER or MEMORY are converted to a UINT and then to a text before printing. New line characters are not appended to strings.

Not all printers support direct text printing. Printers sold as "GDI Only" do not include built in fonts and will not work with the WRITEPRINTER command. For "GDI Only" printers use the PRINTWINDOW command instead.

See Also: [OPENPRINTER](#), [ENDPAGE](#), [CLOSEPRINTER](#), [PRINTWINDOW](#)

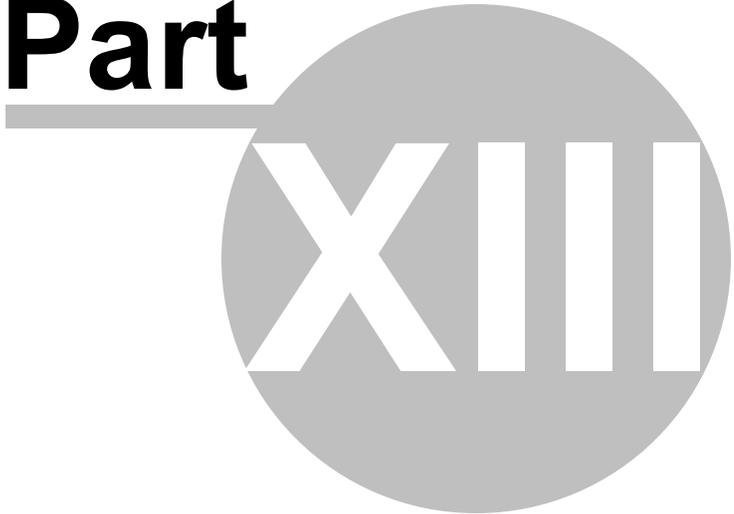
Example usage

```
prtname = GETDEFAULTPRINTER
hPrinter = OPENPRINTER(prtname, "Job 1", "TEXT")
IF hPrinter
    WRITEPRINTER hPrinter, "This is line 1\n"
    CLOSEPRINTER hPrinter
ENDIF
```


Appendix - User's Guide

Part

XIII



13 Appendix - User's Guide

13.1 A. Compiler preprocessor reference

The preprocessor of the compiler handles such things as constants, include files, and conditional compiling. This document contains the current preprocessor commands as of version 2.0. Other commands are built in to the language and covered in the main help files.

`/* */`

Beginning and end of a block comment

`,`

Beginning of a line comment

`_asm`

Begins an inline assembly block

`_endasm`

Ends an inline assembly block

`$command` declaration

Marks a subroutine as a command instead of a function. For internal use and command paks only

`AUTODEFINE "ON" or "OFF"`

Controls auto definition of variables by assignment. If "OFF" then all variables need to be defined before use. Defaults to "ON"

`$ASM`

Begins an inline assembly block Alias for `_asm`.

`$DEFINE conditional_identifier`

Defines a conditional identifier. The following operators may be used: `=`, `==`, `<>`, `!=`, `+`, `-`, `*`, `/`, `%`, `|`, `OR`, `||` (xor), `&`, `&&`, `AND`, `<<`, `>>`, `!`, `~`, `^`, `>`, `>=`, `<`, `<=`.

`#DEFINE conditional_identifier`

Defines a conditional identifier. (same as `$DEFINE`)

`$ELSE`

False conditional block start

`$ELIF`

Provides additional sections (ElseIf) in a conditional compiling block

`$ELIFDEF`

Provides additional sections (ElseIfDefined) in a conditional compiling block

\$ELIFNDEF

Provides additional sections (ElseIfNotDefined) in a conditional compiling block

\$EMIT instruction

Inserts the instruction in an inline assembly block.

\$END

Marks the end of file, any text following it will be ignored..

\$ENDASM

Ends an inline assembly block Alias for `__endasm`.

\$ENDIF

Ends a conditional compiling block

\$ENDREGION

Defines the end of a block of assembly code in the IDE that can be folded out of view. NOTE:
Not currently implemented

\$ERROR "text"

When encountered, prints the "text" in the *Build* pane of the IDE. Causes the compile process to stop.

\$IF expression

Begins a conditional compiling block

\$IFDEF conditional_identifier

Begins a conditional compiling block

\$IFNDEF conditional_identifier

Begins a conditional compiling block

\$INCLUDE "filename"

Include headers and source files into the current compile

\$MACRO

Directive for creating macros. See discussion in [Language/Macros](#) section.

\$MAIN

Marks the beginning execution point of the program. Used in project mode compiling.

\$OPTION "value"

Changes the behavior of the compiler depended on 'value'.

For currently defined options see [Language/Compiler Options/\\$Options keyword](#).

\$REGION

Defines the beginning of a block of assembly code in the IDE that can be folded out of view.
NOTE: Not currently implemented

\$THREAD

Used to create multiple threads within a program. See the discussion in [Language/Threads..](#)

\$TYPEDEF

Alias for [TYPEDEF](#).

\$UNDECLARE name

Removes a previous DECLARE statement from the current compile.

\$UNDEF name

Undefines a condition or constant previously defined by \$DEFINE \#DEFINE

\$USE "filename"

Includes an import library, static library, or object file into the linker stage of the build. \$USE will search for libraries first in folder relative to current file path location, then in default location (installdir/libs/).

```
$use "mylib.lib" ' if exists CurrentFileDir\mylib.lib - use it, else do default
$use "libraries\mylib.lib" ' if exists CurrentFileDir\libraries\mylib.lib - use it, e
$use "..\libraries\mylib.lib" ' if exists CurrentFileParentDir\libraries\mylib.lib -
```

\$WARNING "text"

When encountered, prints the "text" in the *Build* pane of the IDE. The compile process continues.

CONST name = value

Defines a compile time constant

'ENDREGION

Defines the end of a block of code in the IDE that can be folded out of view.

EXPORT function_name

Marks a subroutine as exported in a DLL build.

EXTERN name as type

DECLARE **EXTERN** declaration

Creates a linker reference to an external global variable or subroutine.

GLOBAL function_name

Marks a subroutine as globally visible to other object files

PROJECTGLOBAL "on"/"off"

Marks a subroutine as globally visible to other object files

'REGION

Defines the beginning of a block of code in the IDE that can be folded out of view.

SETID "name", value
Defines a compile time @ constant.

SET_INTERFACE variable, interface name
Sets the compile time interface of a COMREF variable

SETTYPE variable, type name
Sets the compile time type of a POINTER

13.2 B. Message variables, ID's and constants

These are most of the predefined message variables, message ID's and general constants. Any windows message ID can be received by your handler, only the ones used the in the majority of programs are predefined.

Message variables

@MESSAGE / @CLASS
Contains the message ID when your window/dialog procedure is called.

@WPARAM / @CODE
Contains further information about the message. Equivalent to wParam in Windows

@LPARAM / @QUAL
Contains further information about the message. Equivalent to lParam in Windows

@MOUSEX
The mouse X position in the window when the message was sent

@MOUSEY
The mouse Y position in the window when the message was sent

@MENUNUM
Contains the menu number when a @IDMENUPICK message is sent

@CONTROLID
Contains the control ID

@HITWINDOW
Contains a pointer to the window that received the message

@NOTIFYCODE
Contains the notification code sent by a control.

Message ID's

These are the predefined message ID's that will be received by your window/dialog handler.

@IDLBUTTONDBLCLK

Left mouse button was double clicked

@IDLBUTTONUP

Left mouse button was released

@IDRBUTTONUP

Right mouse button was released

@IDMOUSEMOVE

Someone moved the mouse

@IDLBUTTONDN

Left mouse button was pressed

@IDRBUTTONDN

Right mouse button was pressed

@IDRBUTTONDBLCLK

Right mouse button was double clicked

@IDCONTROL

A control was clicked

@IDCLOSEWINDOW

Sent when the close window button or 'close' from the system menu selected

@IDMENUPICK

A menu item was selected. Check @MENUNUM for the ID

@IDMENUINIT

Sent when a menu is about to be displayed. Use CHECKMENUITEM and ENABLEMENUITEM in response to this message to modify the appearance of the menu before it is shown to the user.

@IDMOVE

Sent when a window/dialog has been moved

@IDMOVING

Sent while a window/dialog is moving

@IDSIZE

The window/dialog is sizing or has been resized. This message is a combination of @IDSIZING and @IDSIZECHANGED

@IDSIZECHANGED

The window/dialog size has changed

@IDSIZING

The window/dialog is being resized

@IDHSCROLL

A horizontal scroll bar was clicked. Check @WPARAM for further information

@IDVSCROLL

A vertical scroll bar was clicked. Check @WPARAM for further information

@IDINITDIALOG

A Dialog is about to be displayed

@IDCANCEL

System dialogs return this to DOMODAL when the CANCEL button is pressed

@IDCHAR

A key was pressed and released on the keyboard. Check @WPARAM and @LPARAM for the ASCII value and the raw scan code

@IDKEYDOWN

A key was pressed on the keyboard. . Check @WPARAM and @LPARAM for the ASCII value and the raw scan code

@IDKEYUP

A key was released on the keyboard. . Check @WPARAM and @LPARAM for the ASCII value and the raw scan code

@IDTIMER

Sent when the timer elapses. See STARTTIMER in the user guide

@IDPAINT

Sent when the window needs refreshing and @NOAUTODRAW was used to create the window.

@IDCREATE

Sent when the window is first created but before it is displayed

@IDBEFORENAV

Sent when the embedded browser is about to navigate to a page

@IDNAVCOMPLETE

Sent when the embedded browser has finished loading a page.

@IDSTATUSTEXTUPDATE

Sent when the browser has updates status text to display

@IDERASEBACKGROUND

Sent when the background of a window needs to be painted. The window background is shown when transparent objects, such as toolbars with the @TBFLAT style, are visible.

@IDDESTROY

Sent when the window is about to be destroyed but before it is deleted.

LOADIMAGE constants**@IMGBITMAP**

Loads a bitmap from a file or resource

@IMGICON

Loads an icon from a file or resource

@IMGCURSOR

Loads a cursor from a file or resource

@IMGEMF

Loads an enhanced meta file (EMF) from a disk file

@IMGSCALABLE

Loads a scalable bitmap, JPG, GIF or TIFF image from a file or resource

@IMGOEM

OR in with @IMGBITMAP, @IMGICON or @IMGCURSOR to load an OEM (system) image.

@IMGMAPCOLORS

OR in with @IMGBITMAP to have the system search the bitmap color table and map shades of gray to standard system 3D colors.

SHOWWINDOW constants**@SWRESTORE**

Restore the window to its previous state

@SWMINIMIZED

Minimize the window

@SWMAXIMIZED

Maximize the window

@SWHIDE

Hides the window

@SWSHOW

Shows the window

Scrollbar messages returned in @WPARAM**@SBLEFT**

Scroll to the far left.

@SBENDSCROLL

End scroll.

@SBLINELEFT

Scroll one line left

@SBLINERIGHT

Scroll one line right

@SBPAGELEFT

Scroll one page left

@SBPAGERIGHT

Scroll one page right

@SBRIGHT

Scroll to the far right

@SBTHUMBPOS

Scroll to absolute position. Check @LPARAM for the position, or use GETSCROLLPOSITION

@SBTHUMBTRACK

Drag scroll box to a position. Check @LPARAM for position, or use GETTHUMBPOSITION

@SBBOTTOM

Scroll to the bottom

@SBLINEDOWN

Scroll one line down

@SBLINEUP

Scroll one line up

@SBPAGEDOWN

Scroll one page down

@SBPAGEUP

Scroll one page up

@SBTOP

Scroll to the top

DRAWMODE flags

@TRANSPARENT

Text is drawn using the FRONTPEN color. No background is drawn.

@OPAQUE

Text is drawn using both the FRONTPEN and BACKPEN colors.

Menu creation flags**@MENUDISABLE**

The menu item will be grayed out and not selectable

@MENUCHECK

Shows a checkmark next to the menu item

Button control styles**@CTLBTNBITMAP**

Creates a bitmap button. The buttons caption text is the filename

@CTLBTNMULTI

Creates a multi line button control. Text is automatically word wrapped.

@CTLBTNDEFAULT

Identifies this as the default push button in a dialog

@CTLBTNFLAT

Creates a flat button

Static control styles**@CTLSTCBITMAP**

Create a bitmap static control. The caption text is the filename

@CTLSTCMULTI

Creates a multi line static control

@CTLSTCSIMPLE

Creates a single line static control (default)

Edit / Rich edit control styles**@CTEDITLEFT**

Text is left justified in the edit control

@CTEDITRIGHT

Text is right justified in the edit control

@CTEDITMULTI

Designates a multiline edit control. The default is single-line edit control. When the multiline edit

control is in a dialog box, the default response to pressing the ENTER key is to activate the default button. To use the ENTER key as a carriage return, use the @CTEDITRETURN style. When the multiline edit control is not in a dialog box and the @CTEDITAUTOV style is specified, the edit control shows as many lines as possible and scrolls vertically when the user presses the ENTER key. If you do not specify @CTEDITAUTOV, the edit control shows as many lines as possible and beeps if the user presses the ENTER key when no more lines can be displayed. If you specify the @CTEDITAUTOH style, the multiline edit control automatically scrolls horizontally when the caret goes past the right edge of the control. To start a new line, the user must press the ENTER key. If you do not specify @CTEDITAUTOH, the control automatically wraps words to the beginning of the next line when necessary. A new line is also started if the user presses the ENTER key. The window size determines the position of the word wrap. If the window size changes, the word wrapping position changes and the text is redisplayed. Multiline edit controls can have scroll bars. An edit control with scroll bars processes its own scroll bar messages. Note that edit controls without scroll bars scroll as described in the previous paragraphs and process any scroll messages sent by the parent window.

@CTEDITPASS

Displays an asterisk (*) for each character typed into the edit control.

@CTEDITCENTER

Text is centered within the edit control

@CTEDITRO

The edit control is read only and text can be displayed but not entered

@CTEDITAUTOH

Automatically scrolls text to the right by 10 characters when the user types a character at the end of the line. When the user presses the ENTER key, the control scrolls all text back to position zero.

@CTEDITAUTOV

Automatically scrolls text up one page when the user presses the ENTER key on the last line.

@CTEDITRETURN

Specifies that a carriage return be inserted when the user presses the ENTER key while entering text into a multiline edit control in a dialog box. If you do not specify this style, pressing the ENTER key has the same effect as pressing the dialog box's default push button. This style has no effect on a single-line edit control.

@CTEDITNUMBER

Specified that the edit control only accepts numerals as input (0-9). Any other character will be ignored

List box control styles**@CTLISTEXTENDED**

The user can select multiple items using the SHIFT key and the mouse or special key combinations

@CTLISTMULTI

String selection is toggled each time the user clicks or double-clicks the string. Any number of strings can be selected

@CTLISTSORT

Strings in the list box are sorted alphabetically

@CTLISTSTANDARD

Strings in the list box are sorted alphabetically, and the parent window receives an input message whenever the user clicks or double-clicks a string. The list box contains borders on all sides

@CTLISTNOTIFY

Parent window receives an input message whenever the user clicks or double-clicks a string

@CTLISTTABS

Allows a list box to recognize and expand tab characters when drawing its strings

@CTLISTCOLUMNS

Specifies a multicolumn list box that is scrolled horizontally

Scrollbar control styles**@CTSCROLLHORIZ**

Creates a horizontal scrollbar

@CTSCROLLVERT

Creates a vertical scrollbar

Combo box control styles**@CTCOMBODROPDOWN**

Similar to @CTCOMBOSIMPLE, except that the list box is not displayed unless the user selects an icon next to the edit control.

@CTCOMBODROPLIST

Similar to @CTCOMBODROPDOWN, except that the edit control is replaced by a static text item that displays the current selection in the list box.

@CTCOMBOSIMPLE

Displays the list box at all times. The current selection in the list box is displayed in the edit control.

@CTCOMBOSORT

Automatically sorts strings added to the list box.

@CTCOMBOAUTOHSCROLL

Automatically scrolls the text in an edit control to the right when the user types a character at the

end of the line. If this style is not set, only text that fits within the rectangular boundary is allowed.

List view styles**@LVSALIGNLEFT**

Specifies that items are left-aligned in icon and small icon view.

@LVSALIGNTOP

Specifies that items are aligned with the top of the control in icon and small icon view.

@LVSAUTOARRANGE

Specifies that icons are automatically kept arranged in icon view and small icon view.

@LVSEEDITLABELS

Allows item text to be edited in place. The parent window must process the LVN_ENDLABELEDIT notification message.

@LVSICON

Specifies icon view.

@LVSLIST

Specifies list view.

@LVSNOCOLUMNHEADER

Specifies that a column header is not displayed in report view. By default, columns have headers in report view.

@LVSNOLABELWRAP

Displays item text on a single line in icon view. By default, item text can wrap in icon view.

@LVSNOSCROLL

Disables scrolling. All items must be within the client area.

@LVSNOSORTHEADER

Specifies that column headers do not work like buttons. This style is useful if clicking a column header in report view does not carry out an action, such as sorting.

@LVSREPORT

Specifies report view.

@LVSSHOWSELALWAYS

Always show the selection, if any, even if the control does not have the focus.

@LVSSINGLESEL

Allows only one item at a time to be selected. By default, multiple items can be selected.

@LVSSMALLICON

Specifies small icon view.

@LVSSORTASCENDING

Sorts items based on item text in ascending order.

@LVSSORTDESCENDING

Sorts items based on item text in descending order.

SETLINESTYLE flags**@LSSOLID**

Draw a solid line

@LSDASH

Draw a dashed line

@LSDASHDOT

Draw an alternating dashed dotted line

@LSDASHDOTDOT

Draw an alternating dashed dot dot line

@LSDOT

Draw a dotted line

@LSINSIDE

Draws lines around objects on the inside instead of out.

SETFONT flags**@SFITALIC**

Creates an italicized font

@SFUNDERLINE

Creates an underlined font

@SFSTRIKEOUT

Creates a striked out font

SETCURSOR constants**@CSWAIT**

Displays the hourglass wait cursor

@CSARROW

Displays the standard arrow pointer

@CSCUSTOM

Display a custom cursor loaded with the LOADIMAGE function.

PLAYWAVE constants**@SNDASYNC**

PLAYWAVE returns immediately without waiting for the sound to finish playing.

@SNDSYNC

PLAYWAVE waits for the sound to finish playing

@SNDLOOP

The sound loops continuously until PLAYWAVE is called with a filename of "" or another sound is started. This must also be combined with @SNDASYNC

@SNDNOSTOP

If another sound is playing, PLAYWAVE returns without playing the new sound.

RASTERMODE flags**@RMBLACK**

Pixel is always black

@RMWHITE

Pixel is always white

@RMNOP

Pixel remains unchanged

@RMNOT

Pixel is the inverse of the screen color

@RMCOPYPEN

Pixel is pen color

@RMNOTCOPYPEN

Pixel is the inverse of the pen color

@RMMERGEPENNOT

Pixel is a combination of the pen color and the inverse of the screen color (final pixel = (NOT screen pixel) OR pen).

@RMMASKPENNOT

Pixel is a combination of the colors common to both the pen and the inverse of the screen (final pixel = (NOT screen pixel) AND pen).

@RMMERGENOTPEN

Pixel is a combination of the screen color and the inverse of the pen color (final pixel = (NOT pen) OR screen pixel).

@RMMASKNOTPEN

Pixel is a combination of the colors common to both the screen and the inverse of the pen (final pixel = (NOT pen) AND screen pixel).

@RMMERGE PEN

Pixel is a combination of the pen color and the screen color (final pixel = pen OR screen pixel).

@RMNOTMERGE PEN

Pixel is the inverse of the RMMERGE PEN color (final pixel = NOT(pen OR screen pixel)).

@RMMASK PEN

Pixel is a combination of the colors common to both the pen and the screen (final pixel = pen AND screen pixel).

@RMNOTMASK PEN

Pixel is the inverse of the RMMASK PEN color (final pixel = NOT(pen AND screen pixel)).

@RMXOR PEN

Pixel is a combination of the colors that are in the pen or in the screen, but not in both (final pixel = pen XOR screen pixel).

@RMNOTXOR PEN

Pixel is the inverse of the RMXOR PEN color (final pixel = NOT(pen XOR screen pixel)).

Notification codes

Edit Controls

@ENKILLFOCUS

Control has lost input focus

@ENSETFOCUS

Control has been given the input focus

@ENERRSPACE

Control could not complete an operation because there was not enough memory available

@ENMAXTEXT

While inserting text, the user has exceeded the specified number of characters for the edit control. Insertion has been truncated. This message is also sent either when an edit control does not have the @CTEDITAUTOH style and the number of characters to be inserted exceeds the width of the edit control or when an edit control does not have the @CTEDITAUTOV style and the total number of lines to be inserted exceeds the height of the edit control.

@ENUPDATE

The contents of the control are about to change

@ENCHANGE

The contents of the control have changed.

@ENHSCROLL

The user has clicked the edit control's horizontal scroll bar. Windows sends this message before updating the control.

@ENVSCROLL

The user has clicked the edit control's vertical scroll bar. Windows sends this message before updating the control.

Rich edit controls**@ENKILLFOCUS**

Control has lost input focus

@ENSETFOCUS

Control has been given the input focus

@ENERRSPACE

Control could not complete an operation because there was not enough memory available

@ENMAXTEXT

The limit set by @RTSETLIMITTEXT has been reached,

@ENUPDATE

The contents of the control are about to change
Must be enabled with @RTSETEVENTMASK

@ENSELCHANGE

The current selection has changed.
Must be enabled with @RTSETEVENTMASK

@ENCHANGE

The contents of the control have changed.
Must be enabled with @RTSETEVENTMASK

@ENHSCROLL

The user has clicked the edit control's horizontal scroll bar. Windows sends this message before updating the control.
Must be enabled with @RTSETEVENTMASK

@ENVSCROLL

The user has clicked the edit control's vertical scroll bar. Windows sends this message before updating the control.
Must be enabled with @RTSETEVENTMASK

@ENREQUESTRESIZE

The control's contents are either smaller or larger than the control's window size.
Must be enabled with @RTSETEVENTMASK

List view controls**@NMCLICK**

User has left clicked in the control

@NMDBLCLK

User has double clicked in the control

@NMKILLFOCUS

The control has lost the input focus

@NMSETFOCUS

The control has received the input focus

@NMRCLICK

User has right clicked in the control

@LVNCOLUMNCLICK

Indicates that the user clicked a column header in report view.

@LPARAM contains a memory handle to a **NMLISTVIEW** data type.

@LVNKEYDOWN

Signals a keyboard event

@LPARAM contains a memory handle to a **LVKEYDOWN** data type

@LVNBEGINLABELEDIT

Signals the start of in-place label editing

@LVNENDLABELEDIT

Signals the end of label editing

@LVNITEMCHANGED

Indicates that an item has changed.

@LPARAM contains a memory handle to a **NMLISTVIEW** data type.

@LVNITEMCHANGING

Indicates that an item is in the process of changing

@LPARAM contains a memory handle to a **NMLISTVIEW** data type.

@LVNINSERTITEM

Signals the insertion of a new list view item.

@LPARAM contains a memory handle to a **NMLISTVIEW** data type.

@LVNDELETEITEM

Signals the deletion of a specific item

@LPARAM contains a memory handle to a **NMLISTVIEW** data type.

List box controls

@LBNDLCLK

The user double-clicks an item in the list box.

@LBNERRSPACE

The list box cannot allocate enough memory to fulfill a request.

@LBNKILLFOCUS

The list box loses the keyboard focus.

@LBNSETFOCUS

The list box receives the keyboard focus.

@LBNSELCHANGE

The selection in a list box is about to change.

@LBNSELCANCEL

The user cancels the selection of an item in the list box.

Combo box controls**@CBNDLCLICK**

Indicates the user has double-clicked a list item in a simple combo box.

@CBNERRSPACE

Indicates the combo box cannot allocate enough memory to carry out a request, such as adding a list item.

@CBNKILLFOCUS

Indicates the combo box is about to lose the input focus.

@CBNSETFOCUS

Indicates the combo box has received the input focus.

@CBNDROPDOWN

Indicates the list in a drop-down combo box or drop-down list box is about to open.

@CBNCLOSEUP

Indicates the list in a drop-down combo box or drop-down list box is about to close.

@CBNEDITCHANGE

Indicates the user has changed the text in the edit control of a simple or drop-down combo box. This notification message is sent after the altered text is displayed.

@CBNEDITUPATE

Indicates the user has changed the text in the edit control of a simple or drop-down combo box. This notification message is sent before the altered text is displayed.

@CBNSELCHANGE

Indicates the current selection has changed.

@CBNSELENDOK

Indicates that the selection made drop down list, while it was dropped down, should be accepted.

@CBNSELENDNCANCEL

Indicates that the selection made in the drop down list, while it was dropped down, should be ignored.

13.3 C. Virtual key codes

The following table shows the hexadecimal values and keyboard equivalents for the virtual-key codes used by the Microsoft Windows operating system. The codes are listed in numeric order.

Value (hexadecimal)	Mouse or keyboard equivalent
01	Left mouse button
02	Right mouse button
03	Control-break processing
04	Middle mouse button (three-button mouse)
05-07	Undefined
08	BACKSPACE key
09	TAB key
0A-0B	Undefined
0C	CLEAR key
0D	ENTER key
0E-0F	Undefined
10	SHIFT key
11	CTRL key
12	ALT key
13	PAUSE key
14	CAPS LOCK key
15-19	Reserved for Kanji systems
1A	Undefined
1B	ESC key
1C-1F	Reserved for Kanji systems
20	SPACEBAR
21	PAGE UP key
22	PAGE DOWN key
23	END key
24	HOME key
25	LEFT ARROW key

26	UP ARROW key
27	RIGHT ARROW key
28	DOWN ARROW key
29	SELECT key
2A	Original equipment manufacturer (OEM) specific
2B	EXECUTE key
2C	PRINT SCREEN key for Windows 3.0 and later
2D	INS key
2E	DEL key
2F	HELP key
30	0 key
31	1 key
32	2 key
33	3 key
34	4 key
35	5 key
36	6 key
37	7 key
38	8 key
39	9 key
3A-40	Undefined
41	A key
42	B key
43	C key
44	D key
45	E key
46	F key
47	G key
48	H key
49	I key
4A	J key
4B	K key
4C	L key
4D	M key
4E	N key
4F	O key
50	P key
51	Q key
52	R key

53	S key
54	T key
55	U key
56	V key
57	W key
58	X key
59	Y key
5A	Z key
5B	Left Windows key (Microsoft Natural Keyboard)
5C	Right Windows key (Microsoft Natural Keyboard)
5D	Applications key (Microsoft Natural Keyboard)
5E-5F	Undefined
60	Numeric keypad 0 key
61	Numeric keypad 1 key
62	Numeric keypad 2 key
63	Numeric keypad 3 key
64	Numeric keypad 4 key
65	Numeric keypad 5 key
66	Numeric keypad 6 key
67	Numeric keypad 7 key
68	Numeric keypad 8 key
69	Numeric keypad 9 key
6A	Multiply key
6B	Add key
6C	Separator key
6D	Subtract key
6E	Decimal key
6F	Divide key
70	F1 key
71	F2 key
72	F3 key
73	F4 key
74	F5 key
75	F6 key
76	F7 key
77	F8 key
78	F9 key
79	F10 key

7A	F11 key
7B	F12 key
7C	F13 key
7D	F14 key
7E	F15 key
7F	F16 key
80H	F17 key
81H	F18 key
82H	F19 key
83H	F20 key
84H	F21 key
85H	F22 key
86H	F23 key
87H	F24 key
88-8F	Unassigned
90	NUM LOCK key
91	SCROLL LOCK key
92-B9	Unassigned
BA-C0	OEM specific
C1-DA	Unassigned
DB-E4	OEM specific
E5	Unassigned
E6	OEM specific
E7-E8	Unassigned
E9-F5	OEM specific
F6	Attn key
F7	CrSel key
F8	ExSel key
F9	Erase EOF key
FA	Play key
FB	Zoom key
FC	Reserved for future use.
FD	PA1 key
FE	Clear key

13.4 D. ASCII table

ASCII character set

The Windows OS uses the Latin-1 character set.

Dec	Hex	Char									
0	00	NUL	64	40	@	128	80	PAD	192	C0	À
1	01	SOH	65	41	A	129	81	HOP	193	C1	Á
2	02	STX	66	42	B	130	82	BPH	194	C2	Â
3	03	ETX	67	43	C	131	83	NBH	195	C3	Ã
4	04	EOT	68	44	D	132	84	IND	196	C4	Ä
5	05	ENQ	69	45	E	133	85	NEL	197	C5	Å
6	06	ACK	70	46	F	134	86	SSA	198	C6	Æ
7	07	BEL	71	47	G	135	87	ESA	199	C7	Ç
8	08	BS	72	48	H	136	88	HTS	200	C8	È
9	09	TAB	73	49	I	137	89	HTJ	201	C9	É
10	0A	LF	74	4A	J	138	8A	VTS	202	CA	Ê
11	0B	VT	75	4B	K	139	8B	PLD	203	CB	Ë
12	0C	FF	76	4C	L	140	8C	PLU	204	CC	Ì
13	0D	CR	77	4D	M	141	8D	RI	205	CD	Í
14	0E	SO	78	4E	N	142	8E	SS2	206	CE	Î
15	0F	SI	79	4F	O	143	8F	SS3	207	CF	Ï
16	10	DLE	80	50	P	144	90	DCS	208	D0	Ð
17	11	DC1	81	51	Q	145	91	PU1	209	D1	Ñ
18	12	DC2	82	52	R	146	92	PU2	210	D2	Ò
19	13	DC3	83	53	S	147	93	STS	211	D3	Ó
20	14	DC4	84	54	T	148	94	CCH	212	D4	Ô
21	15	NAK	85	55	U	149	95	MW	213	D5	{
22	16	SYN	86	56	V	150	96	SPA	214	D6	Ö
23	17	ETB	87	57	W	151	97	EPA	215	D7	×
24	18	CAN	88	58	X	152	98	SOS	216	D8	Ø
25	19	EM	89	59	Y	153	99	SGCI	217	D9	Ù
26	1A	SUB	90	5A	Z	154	9A	SCI	218	DA	Ú
27	1B	ESC	91	5B	[155	9B	CSI	219	DB	Û
28	1C	FS	92	5C	\	156	9C	ST	220	DC	Ü
29	1D	GS	93	5D]	157	9D	OSC	221	DD	Ý
30	1E	RS	94	5E	^	158	9E	PM	222	DE	þ
31	1F	US	95	5F		159	9F	APC	223	DF	ß
32	20	SPC	96	60	`	160	A0	NBS	224	E0	à
33	21	!	97	61	a	161	A1	ı	225	E1	á
34	22	"	98	62	b	162	A2	ç	226	E2	â
35	23	#	99	63	c	163	A3	£	227	E3	ã
36	24	\$	100	64	d	164	A4	¤	228	E4	ä
37	25	%	101	65	e	165	A5	¥	229	E5	å
38	26	&	102	66	f	166	A6		230	E6	æ
39	27	'	103	67	g	167	A7	§	231	E7	ç
40	28	(104	68	h	168	A8	¨	232	E8	è

41	29)	105	69	i	169	A9	©	233	E9	é
42	2A	*	106	6A	j	170	AA	^a	234	EA	ê
43	2B	+	107	6B	k	171	AB	«	235	EB	ë
44	2C	,	108	6C	l	172	AC	¬	236	EC	ì
45	2D	-	109	6D	m	173	AD		237	ED	í
46	2E	.	110	6E	n	174	AE	®	238	EE	î
47	2F	/	111	6F	o	175	AF	¯	239	EF	ï
48	30	0	112	70	p	176	B0	°	240	F0	ð
49	31	1	113	71	q	177	B1	±	241	F1	ñ
50	32	2	114	72	r	178	B2	²	242	F2	ò
51	33	3	115	73	s	179	B3	³	243	F3	ó
52	34	4	116	74	t	180	B4	'	244	F4	ô
53	35	5	117	75	u	181	B5	μ	245	F5	õ
54	36	6	118	76	v	182	B6	¶	246	F6	ö
55	37	7	119	77	w	183	B7	·	247	F7	÷
56	38	8	120	78	x	184	B8	,	248	F8	ø
57	39	9	121	79	y	185	B9	ⁱ	249	F9	ù
58	3A	:	122	7A	z	186	BA	°	250	FA	ú
59	3B	;	123	7B	{	187	BB	»	251	FB	û
60	3C	<	124	7C		188	BC	¼	252	FC	ü
61	3D	=	125	7D	}	189	BD	½	253	FD	ý
62	3E	>	126	7E	~	190	BE	¾	254	FE	þ
63	3F	?	127	7F	DEL	191	BF	¿	255	FF	ÿ

13.5 E. Special Constants

The compiler has the following special constants available. The first four are primarily used for debugging purposes; `__WARNINGFILTER__`, `__CODEPAGE__`, `__FILE__`, `__LINE__`, `__IWVER__`

`__WARNINGFILTER__`

Returns the current warning filter

`__CODEPAGE__`

Returns the current codepage

`__FILE__`

Returns the current file path (a string)

`__LINE__`

Returns the current line number

`__IWVER__`

Returns a number with compiler version.

The format is 0xVVxyyz

where

VV is the major version number

xx is the minor level 1 version

yy is the minor level 2 version

zz is the minor level 3 version

```
print "compiler version: 0x", hex$(__IWVER__)
compiler version: 20000905
'which is for the current compiler version of 2.095
```

Note: Currently it does not return anything but TRUE when used in \$IFDEF statement.

__CASESENSITIVE__

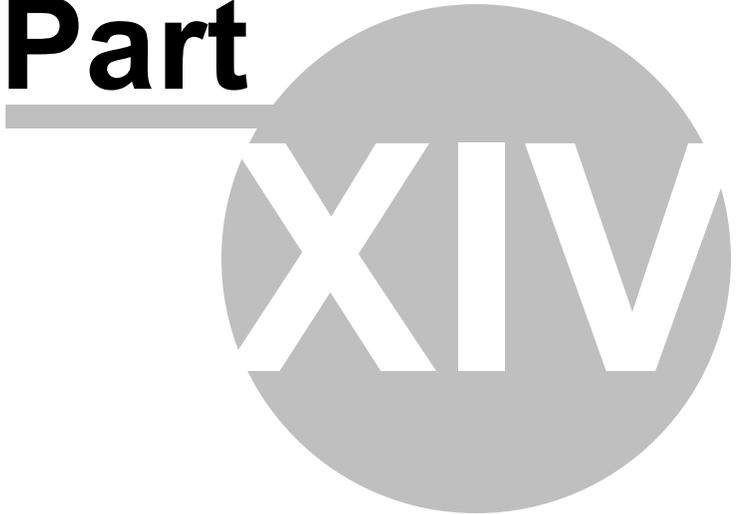
Not a true constant, but will be defined if case sensitive mode has been enabled.

```
print "current error filter: 0x", hex$(__WARNINGFILTER__)
print "current codepage: ", __CODEPAGE__
print "current file: ", __FILE__
print "current line number: ", __LINE__ :
print "compiler version: 0x", hex$(__IWVER__)
```

2D Programming Guide

Part

XIV



14 2D Programming Guide

14.1 Introduction

Introduction

The DirectX 2D command set is an advanced 2D library included with the IWBASIC development environment. Simple to use commands and functions allow creating just about any 2D game or graphics program imaginable.

Main Features:

- Optimized 2D graphics for fast frame rates.
- Supports 8, 16, 24 and 32 bits per pixel screens with triple buffering.
- Supports windowed DirectX mode.
- Built in support for DirectInput devices such as joysticks and game pads.
- Sprite support with pixel perfect collision system.
- Scrolling tile maps for creation of unlimited levels and mazes.
- Fast primitive graphics for drawing lines, pixels, rectangles and text.
- Low level commands for accessing buffer memory directly.
- Interfaces seamlessly with IWBASIC's messaging and window features.

Minimum requirements

- IWBASIC 2.0 or greater
- DirectX 7.0 or greater
- Video card with at least 8MB of ram.
- Windows 95, 98, ME, 2000, XP or Vista

The 2D command set will not work with Windows NT 4.0 since the last version of DirectX for that OS was version 3.0

14.2 Creating and Using Screens

About screens

A screen is the actual display where the sprites, maps and graphics will appear. By bypassing Windows and communicating directly with the video hardware, the DirectX 2D command set achieves fast frame rates, smooth animation and rapid response to user input.

The screen contains two buffers that are accessible by the 2D command set, namely the FrontBuffer and the BackBuffer. The FrontBuffer is what you see on the monitor. It is the memory on the video card being displayed to the user. The BackBuffer can be thought of as a holding area where images are drawn ready to be displayed. By swapping the two buffers, known as *flipping*, the actual drawing can be hidden until ready to be shown. This allows for the illusion of animation

and is how all games are written.

Creating the screen

Before any images, sprites or maps can be drawn the screen must first be created. This is done with one of two commands. [CREATESCREEN](#) creates a full screen exclusive 2D program and [ATTACHSCREEN](#) attaches a 2D screen to a regular window. For now we will concentrate on the full screen kind. Windowed mode will be covered later.

CREATESCREEN is easy to use and you need only specify the width, height and desired bits per pixel (bpp). It is recommended for today's modern video cards to use 16 or 32 bpp as most cards are optimized for that color depth. For width and height use standard screen sizes such as 640x480, 800x600, 1024x768, etc. The larger the screen the more memory is required to show it. CREATESCREEN returns 0 if the screen was successfully opened and < 0 on error. Example of creating a screen:

```
IF CREATESCREEN( 800, 600, 32) < 0
    MESSAGEBOX 0,"Error creating screen","error"
END
ENDIF
```

In the example, we tested to see if the screen was created and if not, displayed a message box and ended the program. For a real program you could try other screen resolutions if creating the screen fails before giving an error.

Closing the screen

When the program ends you must close the screen to return control of the buffer memory back to Windows. This is done with the [CLOSESCREEN](#) command. Before you close the screen make sure all sprites and maps are freed. You must also close the screen if you wish to change resolutions or bits per pixel.

Buffers and flipping

As mentioned earlier there are two buffers that you will work with in any 2D DirectX program. The *BackBuffer* where you normally draw onto and the *FrontBuffer* that is currently being displayed to the user. To swap the front and back buffers use the [FLIP](#) command. The FLIP command has an optional argument to disable waiting for the monitor to refresh (VSYNC). FLIP returns the current FPS (frames per second) speed. For example:

```
speed = FLIP( TRUE )
```

Will swap the front and back buffers and not wait for VSYNC. By not waiting for the monitor refresh, maximum frame rates can be achieved. There are times, however, where it is beneficial to wait for the monitor refresh. Not every program needs fast frame rates or appear to be real time. Flipping as fast as possible also puts the most strain on the system which means multitasking will be

slower, etc.

The default is to flip and wait for VSYNC so its not necessary to supply an argument if you don't need the faster frame rates, as in:

speed = [FLIP](#)

There is also a third buffer called the *SpriteBuffer* that is never directly shown on the screen or flipped but is used to link with a sprites image surface for drawing operations.

Filling the buffer with a solid color

Use the [FILLSCREEN](#) command to fill a buffer with a color. This is usually the first statement in a game loop to erase any graphics left over from the previous flip. remember that flipping swaps the front and back buffers so the old front buffer becomes the new back buffer. If you don't fill the buffer you will have graphic remnants left over from the the previous flip.

Here is a complete example that opens a screen, creates a basic loop that waits for a key to close, fills the BackBuffer (default) and handles flipping:

```
IF CREATESCREEN( 800, 600, 32) < 0
    MESSAGEBOX 0,"Error creating screen","error"
    END
ENDIF
DO
    FILLSCREEN RGB(255,0,0)
    WRITETEXT 0, 0, "Press any key to close"
    FLIP
UNTIL GETKEY <> ""
CLOSESCREEN
END
```

The above code is a basic skeleton that forms the basis of all 2D programs.

14.3 Graphic Commands

The 2D command set features many built in high speed graphic drawing functions specifically designed to take the most advantage of DirectX. These primitive drawing commands can be used along side the built in IW BASIC ones. When using an 8 bit screen with any of the drawing commands substitute a palette number for the RGB color in the examples.

Line drawing

Drawing a solid line is done with the [DRAWLINE](#) command. The line is always one pixel wide and can be any color. The line can be drawn on the back buffer (default), the front buffer or the sprite buffer set with the the `SpriteToBuffer` command. Examples:

```
DRAWLINE 0, 0, 100, 200, RGB(255,0,255)
DRAWLINE 0, 10, 100, 350, RGB(0, 0, 200), FrontBuffer
```

A line can be drawn alpha blended into the background using [DRAWALPHALINE](#). The line is always one pixel wide and can be any color. The line can be drawn on either the back buffer (default) or the front buffer. Examples:

```
DRAWALPHALINE 0, 0, 100, 200, RGB(255,0,255), 128
DRAWALPHALINE 0, 10, 100, 350, RGB(0, 0, 200), 200, FrontBuffer
```

An anti-aliased line can be drawn with the [DRAWAALINE](#) command. Anti-aliasing uses a software algorithm to remove the jagged edges normally seen on lines drawn at an angle.

```
DRAWAALINE 0, 0, 100, 200, RGB(255,0,255)
DRAWAALINE 0, 10, 100, 350, RGB(0, 0, 200), FrontBuffer
```

Rectangles

A rectangle outline is drawn with the [DRAWRECT](#) command. The interior of the rectangle is transparent and will show anything underneath.

```
DRAWRECT 0, 0, 180, 200, RGB(255,255,255)
```

Filled rectangles are drawn with the [DRAWFILLEDRECT](#) command.

```
DRAWFILLEDRECT 10,10, 100, 100, RGB(100,100,100)
```

Writing pixels

Writing a pixel in any color can be done with the [WRITEPIXEL](#) command.

```
WRITEPIXEL 0,100, RGB(77,255,255)
```

An alpha blended pixel can be written to a buffer with the [WRITEALPHAPIXEL](#) command

```
WRITEALPHAPIXEL 0, 230, RGB(20,200,20), 110
```

For high speed effects and advanced uses the [WRITEPIXELFAST](#) command uses hand optimized machine language to write the pixel to the buffer in the fastest time possible. [WRITEPIXELFAST](#) requires locking of the buffer ahead of time before use.

```
LOCKBUFFER
  FOR y=0 to 599
    FOR x=0 to 799
      WritePixelFast(x,y,c)
    NEXT x
  NEXT y
UNLOCKBUFFER
```

You must be very careful not to write outside of the screen dimensions. There are no boundary

checks done with WRITEPIXELFAST. Writing outside of the buffer will result in an access violation.

Drawing text

Text can be drawn to a buffer using the [WRITETEXT](#) command. WRITETEXT is significantly faster than using PRINT.

```
WRITETEXT 0, 0, "Press any key to close"
```

The font, color and drawing mode of the text is set using the commands from IWBASIC which will be covered next.

Using IWBASIC's drawing commands.

All of the drawing commands and functions from the IWBASIC core command set can be used with your 2D screens. The built in variables FrontBuffer, BackBuffer and SpriteBuffer substitute for WINDOW variables in any of the drawing commands. For example:

```
'Change the text color, mode and font of the back buffer
FRONTPEN BackBuffer,RGB(255,255,255)
DRAWMODE BackBuffer,@TRANSPARENT
SETFONT BackBuffer,"Courier New",20,400
'Draw a circle to the back buffer
CIRCLE BackBuffer,110,125,50,RGB(255,0,0),RGB(0,0,255)
'Print some text
MOVE BackBuffer, 0, 0
PRINT BackBuffer, "Press any key to close ", "FPS = ", speed
```

When using IWBASIC's drawing commands with 8 bit screens the [PALETTEINDEX](#) function must be used to map a screens palette to the RGB color needed for the command. Example:

```
SETPALETTECOLOR 3,RGB(0,0,255)
SETPALETTECOLOR 4,RGB(255,0,255)
CIRCLE BackBuffer,110,125,50,PALETTEINDEX(3),PALETTEINDEX(4)
```

14.4 Using Sprites

Overview

Sprites are figures or elements on the screen that have the capability of moving independently of one another. At the lowest level a sprite is simply a bitmap image that can be displayed on the screen. The 2D command set supports many drawing modes for displaying the sprite including transparent, alpha blending, animated, rotated and zoomed.

Sprites can be used for displaying static images as well and will be the most used element in all of your 2D programs.

Loading sprites

Before a sprite can be used it must first be loaded either from a disk file or from your programs resources. The [LOADSPRITE](#) function handles loading the sprite and supports DIB and DDB bitmaps, RLE bitmaps, JPEG, as well as any image format supported by Internet Explorer. A screen must be opened before any sprites can be loaded and you must free all sprites with the [FREESPRITE](#) function before you close the screen.

The loaded sprite can contain one or more image *frames* with each frame being the same width and height. Using frames allows for animation and also having multiple sprites in a single file. Using multiple sprites in one file will save on video card memory as each separate sprite file get allocated a DirectX buffer. The buffer can be located in either video ram or system ram. See the drawing mode table for information on when system ram should be used.

The LOADSPRITE command returns a pointer, or handle, to the sprite. This pointer can be passed to any function that requires a sprite as an argument. It is not necessary to pre-define the variable used for the sprite. Example loading of a sprite:

```
bug1 = LOADSPRITE (GETSTARTPATH + "mouth.bmp", 0, 0, 3)
```

The above statement loads a sprite stored in the programs directory from the file mouth.bmp. The sprite contains 3 image frames. The width and height are specified as 0 to allow LOADSPRITE to calculate the width and height. If the file contains just a single image you can omit width, height and frames completely as in:

```
bug2 = LOADSPRITE (GETSTARTPATH + "bug2.bmp")
```

If you know the width and height but the number of frames may vary you can specify just the width and height and the number of frames will be calculated automatically. If your image consists of many rows and columns of frame images you must specify width, height and the number of frames.

If the sprite cannot be loaded the sprite pointer will be equal to NULL (0).

Moving the sprite

Sprites can keep track of their position on the screen internally. To move a sprite use the [MOVESPRITE](#) command. A sprite can be moved outside of the screen space in which case it will simply be ignored when drawing. The coordinates given to the MOVESPRITE command specify the upper left hand corner of the sprite image.

```
MOVESPRITE bug1, 0, 100
```

Rendering the sprite

The sprite is drawn, or rendered, onto either buffer using either the [DRAWSPRITE](#) or [DRAWSPRITEXY](#) commands. DRAWSPRITE uses the internal position set by the MoveSprite command and DRAWSPRITEXY allows specifying the position directly. A sprite can be drawn on either the front or back buffers by using the optional buffer argument. Examples statements:

```
DRAWSPRITE bug1
DRAWSPRITE bug2, FrontBuffer
DRAWSPRITEXY bug1, 100, 35
DRAWSPRITEXY bug2, 100, 35, FrontBuffer
```

Note that when drawing onto the back buffer, which is default, you must use FLIP to actually see the sprite. This is true of anything drawn onto the back buffer. Drawing on the front buffer will show the sprite immediately but it has limited use since the illusion of animation requires flipping.

DRAWSPRITEXY does not use or update the internally stored position and is most useful when a single sprite image needs to be displayed in multiple locations on the screen.

Sprite drawing modes

Sprites can be rendered to the buffer in a variety of ways. The default drawing mode is a straight copy from image to buffer. To change the drawing mode use the [SPRITEDRAWMODE](#) command. The following table lists the supported modes:

Mode	Comments	Where sprite should be loaded
@BLOCKCOPY	The default drawing mode performs a straight image copy	Video ram
@ALPHA	Alpha blends the sprite into the background. Alpha value is set with the SPRITEALPHA command.	System ram
@SCALED	Scales the sprite using the scaling factor set by the SPRITESCALEFACTOR command.	Video ram
@ROTOZOOM	Scales and rotates the sprite using the scale factor and the angle set with the SPRITEANGLE command	System ram
@HFLIP	Draws the sprite flipped horizontally.	Video ram
@VFLIP	Draws the sprite flipped vertically.	Video ram
@TRANS	Renders the sprite omitting the mask color set with SPRITEMASKCOLOR . Also known as the <i>transparency color</i> .	Video ram
@TRANSALPHA	Alpha blends and masks the sprite.	System ram
@TRANSSHADOW	Draws an alpha blended shadow of the sprite. The offset of the shadow is set with SPRITESHADOWOFFSET . The shadow uses the mask color to determine its shape.	System ram*
@TRANSSCALED	Scales and masks the sprite.	Video ram

@TRANSROTOZOOM	Scales, rotates and masks the sprite.	System ram
@TRANSHFLIP	Draws the sprite flipped horizontally with masking.	Video ram
@TRANSVFLIP	Draws the sprite flipped vertically with masking.	Video ram

The most common drawing mode for a sprite will be @TRANS. Alpha blending and rotating sprites are done through software and will generally be slower than the rest of the modes which use the video cards hardware directly. If you're only scaling a sprite and not rotating, then it is better to use @SCALED or @TRANSCALED instead of the @(TRANS)ROTOZOOM modes.

For the @TRANSSHADOW mode the shadow will draw faster if the sprite is located in system memory. However the sprite itself will draw faster if located in video memory. For the best tradeoff load the sprite twice, once in system and once in video.

Sprite animation

Animation of sprites is achieved by changing the image frame of a loaded sprite. A sprite can have any number of frames that make up an animation sequence but it is advised to keep the amount to a minimum as every frame will use some video card memory. Changing the currently displayed frame is done with the [SPRITEFRAME](#) command. Changing of the frames should be timed to the current FPS or otherwise controlled using a window timer.

The following example shows a sprite with 3 image frames changing frames every 1/4 second. By using the FPS returned by the FLIP command you can control the speed of graphic elements easily.

```

speed = 0
Frame = 0
Change = 0
'Create a screen
IF CREATESCREEN( 640, 480, 32) < 0
    MESSAGEBOX 0,"Error creating screen","error"
    END
ENDIF
'Load the sprite
sprite = LoadSprite(GETSTARTPATH + "mouth.bmp",0 ,0 ,3)
IF sprite <> NULL
    'Set the sprites drawing mode to transparent and specify a mask color
    SpriteDrawMode sprite,@TRANS
    SpriteMaskColor sprite,RGB(87,87,87)
    'The main loop
    DO
        FILLSCREEN RGB(0, 0, 255)
        WRITETEXT 0, 0, "Press ESC to close"
        change++
        IF change > (speed / 4)
            frame++
            IF frame > 2 THEN frame = 0
            change = 0
        ENDIF
        'Set the sprites image frame and draw the sprite
        SpriteFrame sprite, frame
        DrawSpriteXY sprite, 320, 240
        speed = FLIP
    
```

```
UNTIL KEYDOWN(1)
  FREESPRITE sprite
ENDIF
CLOSESCREEN
END
```

Creating and drawing on sprites

Blank sprites are created with the [CREATESPRITE](#) command. The special `SpriteBuffer` buffer is used to draw onto a sprites surface by specifying which sprite to use with the [SpriteToBuffer](#) command.

The `CreateSprite` function returns a pointer to a freshly created sprite of the *width* and *height* specified in either system or video memory. The *frames* parameter specifies how many image frames you plan on creating. Calculate this by using total width / frame width. It must work out to be evenly divisible. For example suppose we want frames that are 64 x 64 and a total of 5 frames. The total width would be 5 * 64 or 320 pixels

```
newSprite = CREATESPRITE(320, 64, 5)
```

Once the sprite is created use `SpriteToBuffer` in preparation of drawing on the sprite.

```
SpriteToBuffer(newSprite)
```

Then use any of the 2D or IWBASIC drawing commands to draw on the sprite specifying *SpriteBuffer* as the buffer/window parameter.

```
'Color the whole sprite red
FILLSCREEN RGB(255,0,0), SpriteBuffer
'Draw rectangles on each sprite frame, each frame is 64 pixels wide
FOR frame = 0 TO 4
  DrawRect frame * 64, 0, 64,64, RGB(0,0,255), SpriteBuffer
NEXT frame
```

When using `WritePixelFast` you must lock the `SpriteBuffer` in the same manner you would with the `BackBuffer` or `FrontBuffer`.

You can also directly access a sprites image memory for advanced operations. See the topic [Direct buffer/sprite writing](#) for more details.

Copying sprites

By using the `SpriteToBuffer` command to attach a newly created sprite to the *SpriteBuffer* you can copy another sprites imagery by using the `DrawSprite` or `DrawSpriteXY` commands. The sprite will be rendered to the new sprites buffer using the current drawing mode, scaling, rotation, alpha blending and frame setting of the sprite to copy.

Copying a sprite in this manner has more advantages over a straight bit copy. For example rotation and scaling is normally a display only drawing mode done through a software algorithm and

collision testing always uses the original sprites dimensions and rotation. By copying the sprite its easy to create a scaled rotated image that will properly collision test and allow for faster frame rates.

For example lets say we wanted to copy a sprite rotated 90 degrees and use a different mask color for the new sprite.

```
'Load the source sprite and set the drawing mode, transparent color and display angle.
sourceSprite = LOADSPRITE (GETSTARTPATH+"bug.bmp", 84, 80, 1)
SpriteDrawMode sourceSprite, @TRANSROTOZOOM
SpriteMaskColor sourceSprite, RGB(87, 87, 87)
SpriteAngle sourceSprite, 90 * .01745

'Create a blank sprite, since the source sprite is rotated 90 degree
'the dimensions are reversed
destSprite = CREATESPITE(80, 84, 1)
'Attach the sprite to the SpriteBuffer for drawing onto.
SpriteToBuffer(destSprite)
'Fill the created sprites buffer with our new mask color
FILLSCREEN RGB(255, 255, 255), SpriteBuffer
'Render the source sprite onto the new sprites buffer
DrawSpriteXY sourceSprite, 0, 0, SpriteBuffer
```

For a direct bit copy set the source sprites drawing mode to `@BLOCKCOPY` and copy the frames individually from source to destination.

A complete example of copying sprites can be found in the *sprite_copying.eba* sample.

14.5 Collision Detecting

Collision detecting is a process in which two sprites, or virtual sprites, are compared to see if any of the pixels in either sprite are overlapping. The detection is done by using screen coordinates and can be either a pixel perfect test or a boundary test.

Pixel perfect mode compares the sprites using the transparency mask set with [SpriteMaskColor](#) and will do a boundary check first. The transparency color is ignored while performing the test allowing true collision testing based on the contents of the sprites image. Think of a sprite of a creature with tentacles, with pixel perfect testing a sprite of a rock moving towards the creature would only show as collided when it actually hit one of the tentacles.

A boundary test on the other hand only compares the boundary rectangles of the sprites. The boundary rectangle is the size of one frame of sprite imagery. Boundary tests are appropriate when the accuracy of collision test is not as important as the speed. A pixel perfect test, while being very fast with the Pro 2D command set, still takes more time to compare pixels then it would just comparing rectangles.

The Pro 2D command set has two built in functions for collision detection. [SpriteCollided](#) and

[SpriteCollidedEx](#).

SpriteCollided

The sprites positions on the screen are determined by the internal coordinates set by the [MoveSprite](#) command. The current image frame is determined by the current frame set by the [SpriteFrame](#) command. The sprites must refer to two different sprites. SpriteCollided allows either a pixel perfect test or a boundary test.

```
IF SpriteCollided(goodguy, badguy, TRUE)
    EndGame( )
ENDIF
IF SpriteCollided(rock, badguy, FALSE)
    KillBadGuy( )
ENDIF
```

SpriteCollidedEx

SpriteCollidedEx is a faster version of SpriteCollided and always does a boundary check first and then a pixel perfect test. It requires directly specifying the sprites positions and frames to test. It will also work with a single sprite pointer which can be used for virtual sprite hit testing. By only loading one sprite and drawing it many times on the screen in different positions, you can create virtual sprites. This can be useful when one image contains all of the sprites used in a game or program.

```
IF SpriteCollidedEx(sprite, BALLS[a].x,BALLS[a].y,0, sprite,BALLS[b].x,BALLS[b].y,0)
    'Reverse balls direction
    ...
ENDIF
```

14.6 Scrolling Tile Maps

A map contains the images and definitions for a scrollable tile map, sometimes referred to as 2D maps. A map consists of tile images and a definition file. The definition file contains the locations where each tile should appear in the map. Using maps allows creation of complex levels and backgrounds using a small set of individual images placed in a grid pattern. This saves on video card memory and CPU overhead as copying a set of small images is much faster than copying a very large one to a buffer.

Creating a map is a two step process. First an image file containing all of the tiles is loaded, second the map must be defined either by loading a data file containing the x, y positions of each tile or by manually assigning tile numbers to the map. A screen must be created before using any of the map functions.

Creating the map, loading the tiles

Create the map by using the [NEWMAP](#) function. NEWMAP returns a pointer to the newly created map and operates in the same manner as LOADSPRITE and supports the same image formats. The pointer returned will be used with all of the map functions and commands. Before your screen is closed you must free the map by using the [FREEMAP](#) statement. Each tile in the image file must have the same dimensions.

```
myMap = NEWMAP (GETSTARTPATH+"mapdata.bmp", 64, 64)
```

Each tile in the image file is assigned a number counting from zero. So if your image file has 4 tiles they will be assigned tile numbers 0,1,2,3. After the tile images are loaded the map is considered to be empty until you give the map a size and set tiles into the grid.

Loading an saving definition files

A definition file contains the size of the map and the tile indexes for every position within the map grid. Load a previously created definition file with the [LOADMAPDATA](#) function.

```
IF LOADMAPDATA(myMap, GETSTARTPATH + "level1.dat", TRUE) = FALSE  
    CLOSESCREEN:MESSAGEBOX( 0, "Failed to load level data","Error"):END  
ENDIF
```

LOADMAPDATA will return FALSE if the file could not be opened. The optional scroll wrap parameter specifies whether the map will display as wrapped around when scrolling or just stop when the edges of the defined map are met.

Saving a map definition file is done with the [SAVEMAPDATA](#) function. The widely used extension for a tile position data file is *.dat* but you can use any extension you wish.

```
SAVEMAPDATA myMap, "level1.dat"
```

Creating an empty map and setting tiles

To define the map dimensions instead of loading a data file use the [CREATEMAPDATA](#) function.

```
CREATEMAPDATA myMap, 10, 10, -1, TRUE
```

Would define a map with dimensions of 10x10, filled with empty tile positions, and allowing scroll wrapping. Once the map dimensions are established you place tiles in the map by using the [SETMAPDATA](#) function. SETMAPDATA takes the x and y coordinate of the map and assigns it a tile number from the image file. It is this tile that will be displayed at the position specified when the map is rendered to the buffer.

```
SETMAPDATA myMap, 0, 1, 0
```

Sets tile number 0 at coordinates 0, 1. It is important to remember that when talking about map coordinates that they refer to the tile grid and not pixels. You can retrieve the currently set tile of a position using the [GETMAPDATA](#) function.

Displaying the map

The map is rendered to the buffer using the [DRAWMAP](#) statement. The default drawing mode for the tiles is `@BLOCKCOPY` which is a straight copy from image to buffer. You can also use transparency, or masking, when drawing by setting the drawing mode with the [MAPDRAWMODE](#) statement to `@TRANS`. Change the mask color with [MAPMASKCOLOR](#).

```
MAPDRAWMODE map,@TRANS
MAPMASKCOLOR map,RGB(43,83,199)
DRAWMAP map
```

A map is usually drawn as the first graphical element in your program. Using transparency you can create multiple layers of maps on the screen.

Moving and scrolling the map

A map can be moved to an absolute pixel position in the map with the [MOVEMAP](#) command. The x and y coordinates are specified in pixels and must be in the total pixel width and height of the map. Normally `MOVEMAP` is only used to initially set the displayed position of the map.

```
MOVEMAP myMap, 0, 0
```

Scrolling the map is the whole purpose of a scrolling tile map. Use the [SCROLLMAP](#) statement to move the map in any direction specified by an offset. `SCROLLMAP` accepts a directional parameter which can be one of `@SCROLLUP`, `@SCROLLDOWN`, `@SCROLLLEFT`, or `@SCROLLRIGHT`. The distance is specified in pixels.

```
IF KEYDOWN(DIK_UP)
    ScrollMap( myMap,@SCROLLUP,2 )
ENDIF
IF KEYDOWN(DIK_DOWN)
    ScrollMap( myMap,@SCROLLDOWN,2 )
ENDIF
IF KEYDOWN(DIK_LEFT)
    ScrollMap( myMap,@SCROLLLEFT,2 )
ENDIF
IF KEYDOWN(DIK_RIGHT)
    ScrollMap( myMap,@SCROLLRIGHT,2 )
ENDIF
```

Changing a maps viewport

The view port is a rectangular area that the map is rendered into. This is set by default to the same size as the screen dimensions. You can restrict the area the map renders into by defining the *viewport rectangle*. When using a viewport, the map automatically shifts the map rendering to match the upper left corner of the viewport as position 0,0.

```
DEF vp as WINRECT
vp.top = 64
```

```
vp.bottom = 480-64
vp.left = 64
vp.right = 640-64
SETMAPVIEWPORT myMap, vp
```

Map information functions

tilenum = [GETMAPDATA](#)(map,x,y)

Returns the tile number located at map position x, y.

width = [GETMAPWIDTH](#)(map)

Returns the width of the map in tiles.

height = [GETMAPHEIGHT](#)(map)

Returns the height of the map in tiles.

pwidth = [GETMAPPIXELWIDTH](#)(map)

Returns the total width of the map in pixels.

pheight = [GETMAPPIXELHEIGHT](#)(map)

Returns the total height of the map in pixels.

size = [GETMAPCOUNT](#)(map)

Returns the size of the map in tiles.

14.7 Mouse and Keyboard Input

Getting the mouse position

The current position of the mouse can be read with the [MOUSEX\(\)](#) and [MOUSEY\(\)](#) commands. Both commands return the position in screen coordinates.

```
mx = MOUSEX()
my = MOUSEY()
```

When working with *windowed mode* Windows sends the current client positions of the mouse with the message `@IDMOUSEMOVE` and the positions are contained in the `@MOUSEX` and `@MOUSEY` variables. See the IW BASIC users guide [Messages and message loops](#) for more details.

Reading mouse buttons

The current up/down state of the mouse buttons can be determined with the [MOUSEDOWN](#) function. The MOUSEDOWN function returns TRUE if the specified button is down or FALSE if up. For the argument use 1 for the left button, 2 for the right and 3 for the middle mouse button:

```
IF MOUSEDOWN(1)
    '... Someone's pressing a mouse button
ENDIF
```

Reading the keyboard

The keyboard is read using DirectInput which is initialized whenever a screen is created or attached. DirectInput allows real time polling of the keyboard for quick response to user input by bypassing the Windows messaging system.

To read one or more keys in raw format, meaning testing whether a specific key is currently down on the keyboard, use the [KEYDOWN](#) function. The KEYDOWN function returns TRUE if the key is currently being pressed or FALSE otherwise. KEYDOWN expects one input parameter, the DirectInput scan code of the key to test. Scan codes can be found in [Appendix A](#). Example fragment:

```
CONST DIK_UP = 0xC8 /* UpArrow on arrow keypad */
CONST DIK_DOWN = 0xD0 /* DownArrow on arrow keypad */
...
IF KEYDOWN(DIK_UP) AND py > 0
    py -= speed
ENDIF
IF KEYDOWN(DIK_DOWN) AND py < (height-10)
    py += speed
ENDIF
```

To retrieve the ASCII value of the currently pressed key use the [GETKEY](#) function. GETKEY has no parameters and returns the currently pressed key as a string. If no key is being pressed then an empty string is returned. GETKEY is equivalent to INKEY\$ for console programs. Example fragment:

```
b$ = ""
a$ = GETKEY
IF a$ <> "" THEN b$ = a$
WRITETEXT 0,20,"You Pressed: " + b$
```

To pause your program and wait for a key press use the [WAITKEY](#) statement. WAITKEY takes an optional argument that will pause until a specific key is pressed or by itself waits for any key to be pressed. The key to wait for is specified as a scan code.

```
'Pause and wait for any key
WAITKEY
'Pause and wait for the ESC key
WAITKEY 0x01
```

Flushing the keyboard buffer

Flushing the current keyboard buffer is accomplished using the [FLUSHKEYS](#) command. FLUSHKEYS is used when you want to ignore any keyboard input that might have occurred before a WAITKEY or GETKEY function.

Keyboard input example:

```
'some convenient scancodes
CONST DIK_UP = 0xC8 /* UpArrow on arrow keypad */
CONST DIK_LEFT = 0xCB /* LeftArrow on arrow keypad */
CONST DIK_RIGHT = 0xCD /* RightArrow on arrow keypad */
CONST DIK_DOWN = 0xD0 /* DownArrow on arrow keypad */

width = 640
height = 480
fps = 0
CREATESCREEN width,height,16

px = 0.0f
py = 0.0f
speed = 1
b$ = ""
DO
    FILLSCREEN RGB(0,255,0)
    WRITETEXT 0,0,"Press escape to close " + STR$(fps)
    DRAWFILLEDRECT px,py,10,10,RGB(255,0,0)
    IF KEYDOWN(DIK_UP) AND py > 0
        py -= speed
    ENDIF
    IF KEYDOWN(DIK_DOWN) AND py < (height-10)
        py += speed
    ENDIF
    IF KEYDOWN(DIK_LEFT) AND px > 0
        px -= speed
    ENDIF
    IF KEYDOWN(DIK_RIGHT) AND px < (width-10)
        px += speed
    ENDIF

    a$ = GETKEY
    IF a$ <> "" THEN b$ = a$
    WRITETEXT 0,20,"You Pressed: " + b$
    fps = FLIP 1
UNTIL KEYDOWN(0x01)
CLOSESCREEN
END
```

14.8 Joysticks and Gamepads

Using a joystick or gamepad device in your program requires an open screen or manual initialization of DirectInput. The 2D command set supports any number of connected joystick devices and supports a maximum of 32 buttons per device and a maximum of 3 axes per device (X, Y and Z).

Getting the number of devices and their names.

To determine the number of joystick devices attached to the system use the [GETJOYSTICKCOUNT](#) function. Once the number of devices is obtained use a loop to retrieve the names of the devices to present to the user if desired with the [GETJOYSTICKNAME](#) function.

```
joycount = GETJOYSTICKCOUNT
FOR x = 0 TO joycount-1
    name$ = GETJOYSTICKNAME( x )
    ' do something with the name
NEXT x
```

The name returned is usually descriptive such as "2 button, 2 axis joystick" and may include the manufacturers name.

Device types

A joystick devices type can be obtained using the [GETJOYSTICKTYPE](#) function. The returned value will be one of the following constants:

```
@JOYTYPE_UNKNOWN
@JOYTYPE_TRADITIONAL
@JOYTYPE_FLIGHTSTICK
@JOYTYPE_GAMEPAD
@JOYTYPE_RUDDER
@JOYTYPE_WHEEL
@JOYTYPE_HEADTRACKER
```

Use the type to determine if the device is usable in your program

Device settings

Once a device is chosen by the user, or by your program, set up the axis ranges and dead-zones as required for your application. The number of axes supported by the device can be retrieved with the [GETJOYSTICKAXISCOUNT](#) function. Use [GETJOYSTICKBUTTONCOUNT](#) to retrieve the number of buttons the device has.

The range of a joystick axis is set by the [SETJOYSTICKRANGE](#) function. The default range is from -1000 to +1000 and will be the range of values returned when querying the device.

The dead-zone of an axis is set by the [SETJOYSTICKDEADZONE](#) function. A dead-zone is a percentage of the range of movement about the center of the axis where the joystick will report being at the center of its range. The default dead-zone is 10%.

Example of setting up the device:

```
SETJOYSTICKRANGE @XAXIS, -32767, 32767, 1
SETJOYSTICKRANGE @YAXIS, -32767, 32767, 1
SETJOYSTICKDEADZONE @XAXIS, 8.5, 1
SETJOYSTICKDEADZONE @YAXIS, 8.5, 1
```

A digital joystick will report the minimum and maximum for a particular axis.

Reading the device

The current positions of the axes can be retrieved at any time using the [JOYX](#), [JOYY](#), and [JOYZ](#) functions. While an axis is in its dead-zone the functions will return 0. For example to read the three axes of the second joystick:

```
posy = JOYY(1)
posx = JOYX(1)
posz = JOYZ(1)
```

Use [JOYDOWN](#) to determine if a particular button on the device is currently being pressed. JOYDOWN returns 1 if the button is down or 0 if up. Button numbers are zero based so to check the 4th button on device number 1:

```
IF JOYDOWN( 3, 1) = 1 THEN fire_missile = TRUE
```

14.9 Using 8 bpp Screens

When using 8 bpp screen, also known as palletized screens, there are some special functions for setting and using colors. Unlike 16,24 and 32 bpp true color screen the 8 bpp screen uses a palette to store colors and a palette index to specify the color in drawing functions.

The palette

The palette consists of 256 entries (0 - 255) that can store an RGB color. The palette may be set manually or loaded from a bitmap image.

To load a palette from a bitmap image use the [LOADPALETTE](#) command. LOADPALETTE only loads the palette data from the image, not the image itself.

```
LOADPALETTE GETSTARTPATH+"pal.bmp"
```

In order for sprites to display in the correct colors on an 8 bpp screen the screens palette must be identical to the sprites. Only 256 color bitmaps can be used properly as sprite images.

Setting a palette color can be done using the [SETPALETTECOLOR](#) command.

```
SETPALETTECOLOR 2, RGB(255,255,0)
```

Retrieving the color stored in a palette is done with [GETPALETTECOLOR](#)

```
col = GETPALETTECOLOR(2)
```

The entire 255 entry palette can be filled with a single color using [FILLPALETTE](#)

```
FILLPALETTE RGB(255,255,255)
```

Fading the screen

One of the neat tricks to do with a palletized screen is create a smooth fade to color effect by changing palette values on the fly over time. Use the [FADEPALETTE](#) command to achieve this.

```
CREATESCREEN 640,480,8
SETPALETTECOLOR 2,RGB(255,0,0)
DO
  FILLSCREEN 2
  WRITETEXT 0,0,"Press ESC to fade and exit"
  FLIP
UNTIL KEYDOWN(1)
'4 second FADE to black. 0 = RGB(0,0,0).
FADEPALETTE 0,50
CLOSESCREEN
END
```

The time value is specified in 2/25th of a second (0.08). So to specify 8 seconds the value would be $8/0.08 = 100$. Some common values are:

13 = about 1 second

25 = 2 seconds

50 = 4 seconds

100 = 8 seconds

Use with drawing commands

The 2D drawing commands such as DRAWLINE and DRAWRECT accept a palette index directly when using 8 bpp screens.

```
SETPALETTECOLOR 1, RGB(255,0,0)
DRAWLINE 0, 0, 100, 100, 1
```

The IW BASIC drawing commands require the conversion function [PALETTEINDEX](#) to properly work with 8 bpp screens

```
CIRCLE BackBuffer,110,125,50,PALETTEINDEX(3),PALETTEINDEX(4)
```

Notes

Alpha blending is not possible with 8 bpp screens in as much drawing functions that use an alpha value will fail. These include DrawAlphaLine, WriteAlphaPixel and the drawing modes @ALPHA, @TRANSALPHA, and @TRANSSHADOW for sprites.

When using direct buffer writing remember that an 8bpp screen uses a single byte for each pixel and each byte is a palette index number, not an RGB color.

14.10 Windowed mode

The 2D command set supports a windowed mode DirectX screen by using the [ATTACHSCREEN](#) function. Attaching a screen to a regular IW BASIC window allows running a 2D program on the desktop without taking up full control of the screen.

When opening the window to be used as the base of the screen you should always specify `@NOAUTODRAW` as one of the style flags in the `OPENWINDOW` statement. This prevents the IW BASIC window from trying to overwrite the attached screen

```
DEF win as WINDOW
width = 640:height = 480
OPENWINDOW win,0,0,width,height,@NOAUTODRAW|@SIZE,0,"Windowed 2D screen",&myhandler
IF (ATTACHSCREEN(win,width,height,TRUE) < 0)
    MESSAGEBOX win,"Couldn't create DirectX window","Error"
    CLOSEWINDOW win
END
ENDIF
```

The optional *bStretch* parameter of the `ATTACHSCREEN` function specifies how the 2D system should handle copying the back buffer to the windowed screen. If `TRUE` it will stretch to fit the screen allowing the window to be resized and the screen will be automatically scaled to fit the client area of the window. If `FALSE` then the screen is copied as sized. If your not using the stretch to fit feature then use the IW BASIC `GETCLIENTSIZE` command to determine the exact dimensions of the screen to use.

The attached screen will have the same bpp as the users Windows screen. The `FLIP` command has to do a bit copy from the back buffer to the window instead of a hardware flip. Because of this a windowed 2D program will be slower than an identical full screen version.

Closing the screen

You must close the screen with the [CLOSESCREEN](#) command before using `CLOSEWINDOW`. Failing to do so will result in strange behavior such as the program sticking in memory or an access violation.

```
WAITUNTIL run=0
CLOSESCREEN
CLOSEWINDOW win
END
```

Notes

Only one windowed screen at a time is allowed to be attached.

A complete windowed 2D example can be seen in the `dx_windowed.iwb` sample

14.11 Direct Buffer/Sprite Writing

Direct buffer writing, also known as direct memory access, gives you the power to directly modify a buffers contents using a memory pointer. This is an advanced feature and should not be attempted unless you really understand everything presented here. Writing beyond the confines of a DirectX buffer can and will crash your program, or destabilize the system.

A DirectX buffer is an area of memory either on the video card or in system ram. The benefits of direct buffer writing include increased speed, creating custom drawing commands, and special effects.

The buffer is comprised of scan lines equal to the vertical size of the created screen. The length of each scan line in bytes is called the buffer *pitch*. The pitch will be the number of bytes required to store the horizontal pixel information plus some overhead used by DirectX. The length in bytes of a pixel is determined by the screens bits per pixel (bpp).

8 bpp = 1 byte per pixel
16 bpp = 2 bytes per pixel
24 bpp = 3 bytes per pixel
32 bpp = 4 bytes per pixel

I mentioned the overhead used by DirectX because its important to remember that the length of one scan line is not simply the horizontal size * bytes per pixel, you must have the actual pitch to perform direct buffer writing.

Before writing or reading to a buffer can happen it must be locked first. Use the [LOCKBUFFER](#) command to lock the buffer and the [UNLOCKBUFFER](#) command when you are finished with direct buffer writing. To obtain the pitch of the buffer use [GETBUFFERPITCH](#). Use [GETBUFFERPOINTER](#) to retrieve the starting memory location of the buffer. The buffer pointer must be DEFINed as type POINTER ahead of time.

```
DEF pBuffer as POINTER
LOCKBUFFER
pitch = GETBUFFERPITCH
pBuffer = GETBUFFERPOINTER
```

Do not store the pitch and buffer between calls to lock the buffer. DirectX is free to change buffer geometry any time it sees fit. Once the pitch and buffer pointer is obtained a simple calculation is used to determine the location of the pixel in question. You can use direct pointer math, or assign to a temporary pointer.

$pTemp = pBuffer + (x * \text{bytes per pixel}) + (y * \text{pitch})$

Now comes the question of what to write to the memory location. DirectX stores pixel formats differently depending on the video card. Never assume a particular format, such as BGR, is used. To correctly determine a color value to store use the [RGBToScreen](#) function. RGBToScreen takes an RGB color and converts it to the pixel format used by the current screen. It is not needed for 8

bpp screens which only store a palette index as a byte.

Once a color is correctly converted use pointer dereferencing to store the pixel into the buffer. The type casting of the pointer depends on the bpp of the screen. 32bpp = UINT, 16bpp = WORD, 8bpp = CHAR. 24bpp is a bit more difficult which we will cover separately.

```
'Write a pixel to a 32 bpp screen.
#<UINT>pTemp = RGBToScreen(255,0,0)

'Write a pixel to a 16 bpp screen
#<WORD>pTemp = RGBToScreen(0,0,255)

'Write a pixel to an 8 bit screen. Uses a palette index instead
#<CHAR>pTemp = 1
```

For a 24 bpp screen you must use two writes instead of one. First a WORD sized write and then a CHAR sized write. Because of this a 24bpp screen will always be the lowest performer of all the screen modes.

```
'Write a pixel to a 24 bpp screen
col = RGBToScreen(255,0,255)
#<WORD>pTemp = col & 0xFFFF
pTemp += 2
#<CHAR>pTemp = (col >> 16) & 0x00FF
```

Complete example of direct buffer writing:

```
CONST width = 640
CONST height = 480
IF CREATESCREEN(width,height,16) < 0
    MESSAGEBOX 0, "Error: Couldn't create a screen\nBe sure you have DirectX 7.0 or g
    END
ENDIF
'just to show the speed
DEF fps as INT
'The pointer to the buffer
DEF pBuffer as POINTER

DO
    LOCKBUFFER
        pitch = GetBufferPitch
        pBuffer = GetBufferPointer
        FOR y = 0 to height-1
            FOR x = 0 to width-1
                #<WORD>pBuffer[x] = RAND(0xFFFF)
            NEXT x
            pBuffer += pitch
        NEXT Y
    UNLOCKBUFFER
    WRITETEXT 0,0, "Press ESC to close" + " FPS=" + STR$(fps)
    fps = FLIP 1
UNTIL KEYDOWN(0x01)
CLOSESCREEN
END
```

The example above creates a static pattern like you would see on a TV with no input signal. For

the example we didn't use any specific colors but just a random range of values from 0 to 0xFFFF.

The real power of direct buffer writing comes when combined with inline assembly to create very fast drawing routines.

Direct sprite writing

The 2D command set also allows modifying sprites directly. Sprites are also stored on a DirectX buffer and can be locked for direct manipulation of the sprites pixels. [LOCKSPRITE](#), [GETSPRITEPITCH](#), [GETSPRITEPOINTER](#), and [UNLOCKSPRITE](#) commands are used in the same manner as their buffer counterparts:

```
LOCKSPRITE sprite
pitch = GETSPRITEPITCH sprite
pBuffer = GETSPRITEPOINTER sprite
    ... do something with the buffer
UNLOCKSPRITE sprite
```

The bpp for a sprite will be the same as the created screen. As with direct buffer writing you must be sure not to overwrite a sprites buffer.

Creating a blank sprite

To create a blank sprite for direct sprite buffer writing use the [CREATESPRITE](#) function. The CreateSprite function returns a pointer to a freshly created sprite of the *width* and *height* specified. The *frames* parameter specifies how many image frames you plan on creating. Calculate this by using total width / frame width. It must work out to be evenly divisible. For example suppose we want frames that are 64 x 64 and a total of 5 frames. The total width would be 5 * 64 or 320 pixels

```
newSprite = CREATESPRITE(320, 64, 5)
```

Always check the return value of CreateSprite before attempting to lock and write to it.

14.12 Alphabetical Command Reference

[ATTACHSCREEN](#)
[CLOSESCREEN](#)
[CREATEMAPDATA](#)
[CREATESCREEN](#)
[CreateSprite](#)
[DrawAALine](#)
[DrawAlphaLine](#)
[DrawFilledRect](#)
[DrawLine](#)
[DRAWMAP](#)
[DrawRect](#)

[DrawSprite](#)
[DrawSpriteXY](#)
[FADEPALETTE](#)
[FILLPALETTE](#)
[FILLSCREEN](#)
[FLIP](#)
[FLUSHKEYS](#)
[FREEMAP](#)
[FreeSprite](#)
[GetBufferHeight](#)
[GetBufferPitch](#)
[GetBufferPointer](#)
[GetBufferWidth](#)
[GETJOYSTICKAXISCOUNT](#)
[GETJOYSTICKBUTTONCOUNT](#)
[GETJOYSTICKCOUNT](#)
[GETJOYSTICKNAME](#)
[GETJOYSTICKTYPE](#)
[GETKEY](#)
[GETMAPCOUNT](#)
[GETMAPDATA](#)
[GETMAPHEIGHT](#)
[GETMAPPIXELHEIGHT](#)
[GETMAPPIXELWIDTH](#)
[GETMAPWIDTH](#)
[GETPALETTECOLOR](#)
[GetSpriteDelay](#)
[GetSpriteFrames](#)
[GetSpriteHeight](#)
[GetSpritePitch](#)
[GetSpritePointer](#)
[GetSpriteState](#)
[GetSpriteType](#)
[GetSpriteVelX](#)
[GetSpriteVelY](#)
[GetSpriteWidth](#)
[JOYDOWN](#)
[JOYX](#)
[JOYY](#)
[JOYZ](#)
[KEYDOWN](#)
[LOADMAPDATA](#)
[LOADPALETTE](#)
[LoadSprite](#)

[LOCKBUFFER](#)
[LOCKSPRITE](#)
[MAPDRAWMODE](#)
[MAPMASKCOLOR](#)
[MOUSEDOWN](#)
[MOUSEX](#)
[MOUSEY](#)
[MOVEMAP](#)
[MoveSprite](#)
[NEWMAP](#)
[PALETTEINDEX](#)
[ReadPixel](#)
[RGBToScreen](#)
[SAVEMAPDATA](#)
[SCROLLMAP](#)
[SETJOYSTICKDEADZONE](#)
[SETJOYSTICKRANGE](#)
[SETMAPDATA](#)
[SETMAPVIEWPORT](#)
[SETPALETTECOLOR](#)
[SetSpriteDelay](#)
[SetSpriteState](#)
[SetSpriteType](#)
[SetSpriteVelX](#)
[SetSpriteVelY](#)
[SpriteAlpha](#)
[SpriteAngle](#)
[SpriteCollided](#)
[SpriteCollidedEx](#)
[SpriteDrawMode](#)
[SpriteFrame](#)
[SpriteMaskColor](#)
[SpriteScaleFactor](#)
[SpriteShadowOffset](#)
[SpriteToBuffer](#)
[UNLOCKBUFFER](#)
[UNLOCKSPRITE](#)
[WAITKEY](#)
[WriteAlphaPixel](#)
[WritePixel](#)
[WritePixelFast](#)
[WriteText](#)

14.12.1 ATTACHSCREEN

Syntax

INT = ATTACHSCREEN(win as WINDOW,width as int,height as int,OPT bStretch=0 as INT)

Description

Attaches a DirectX screen to a window to create a non exclusive windowed mode DirectX program.

Parameters

win - Window to attach screen to

width - Width of DirectX screen

height - Height of DirectX screen

bStretch - Optional. 1 to scale screen to window, 0 to use the sizes provided.

Return value

0 if DirectX screen was created successfully and attached to the window. < 0 if creation fails.

Remarks

Only one windowed mode DirectX screen is allowed at a time in your program. If a screen already exists then this function will fail. You must use the CLOSESCREEN command to remove the DirectX screen before closing the window.

See Also: [CLOSESCREEN](#), [CREATESCREEN](#)

Example usage

```
OPENWINDOW win,0,0,width,height,@NOAUTODRAW|@SIZE,0,"Caption",&handler
IF(ATTACHSCREEN(win,width,height,TRUE) < 0)
  MESSAGEBOX win,"Couldn't create DirectX window","Error"
CLOSEWINDOW win
END
ENDIF
```

14.12.2 CLOSESCREEN

Syntax

CLOSESCREEN

Description

Closes a DirectX screen previously created with CREATESCREEN or ATTACHSCREEN

Parameters

None

Return value

None

Remarks

The screen is closed, any allocated buffers are freed, and the primary Windows surface is re-enabled if in full screen mode. You must free all sprites before using this command. DirectX will remain initialized so it is safe to call `CREATESCREEN` or `ATTACHSCREEN` again after using this command.

See Also: [CREATESCREEN](#), [ATTACHSCREEN](#)

Example usage

```

CREATESCREEN 800,600,32
FILLSCREEN RGB(155,155,155)
WRITETEXT 0,0,"Press any key to close"
FLIP
WAITKEY
CLOSESCREEN
END

```

14.12.3 CREATEMAPDATA

Syntax

`INT = CREATEMAPDATA(map as POINTER, width as INT, height as INT, fill_tile as INT, opt scrollwrap = FALSE as INT)`

Description

Creates empty tile data for the map. Initially filled with the tile number specified.

Parameters

map - Handle to a map returned by the `NEWMAP` function.
width - Width of the data, in tiles.
height - height of the data, in tiles.
fill_tile - Tile# to initially fill the blank map with.
scrollwrap - Optional. Specifies a wrap around map when scrolling.

Return value

TRUE is map data could be created, FALSE on failure

Remarks

Free the map and its data with `FREEMAP`. Set tiles in the map with `SETMAPDATA`

See Also: [NEWMAP](#), [FREEMAP](#), [SETMAPDATA](#)

Example usage

```

CREATESCREEN width,height,32
map = NEWMAP(GETSTARTPATH+"mapdata.bmp",64,64)
IF map = NULL
  CLOSESCREEN
  MESSAGEBOX 0,"failed to load map images","error"
END
ENDIF
CREATEMAPDATA map,100,100,0,TRUE
...

```

14.12.4 CREATESCREEN

Syntax

INT = CREATESCREEN(width as int,height as int, BPP as int, opt handler=NULL as UINT)

Description

Creates a full screen exclusive DirectX screen complete with front and back buffers. Triple buffering is automatically used if enough memory is available.

Parameters

width - width of the DirectX screen

height - Height of the DirectX screen

BPP - Bits per pixel, one of 8, 16, 24 or 32.

handler - Optional. Address of a windows handler to pass messages to.

Return value

0 if screen was successfully created and < 0 on failure

Remarks

CREATESCREEN or ATTACHSCREEN must be used successfully before any sprite, map or 2D drawing commands are used. The return value should always be tested since not every screen mode will be available on all video cards. It is safe to assume that the system screen size and bit depth will be available. The optional handler allows using a standard windows handler with your full screen program to process messages sent to the internal window. This allows seamless integration standard event driven code with linear 2D code.

See Also: [CLOSESCREEN](#), [ATTACHSCREEN](#)

Example usage

```
IF CREATESCREEN( 800, 600, 32) < 0
    MESSAGEBOX 0,"Error creating screen","error"
ENDIF
FILLSCREEN 0
WRITETEXT 0, 0, "Press any key to close"
FLIP
WAITKEY
CLOSESCREEN
END
```

14.12.5 CreateSprite

Syntax

POINTER = CreateSprite(width as INT,height as INT,frames as INT, OPT bSystemRam as INT)

Description

Creates a blank sprite for direct buffer writing.

Parameters

width - width of sprite

height - height of sprite

frames - number of frames the image will contain

bSystemRam - Optional. If TRUE then creates the sprite in system memory instead of video memory.

Return value

A pointer to the created sprite.

Remarks

Always check the return value. The frames parameter specifies how many frame your image will have using the total width supplied. It must be equal to $frame\ width * frames = width$ so to specify 5 frames and a frame width of 64 with a height of 64 the function would be CreateSprite (320,64,5)

See Also: [FREESPRITE](#), [LOADSPRITE](#), [LOCKSPRITE](#), [UNLOCKSPRITE](#)

Example usage

```
goodguy = CreateSprite(320,64,5)
```

14.12.6 DrawAALine

Syntax

DrawAALine(x as INT,y as INT,x2 as INT,y2 as INT,col as UINT,opt buffer as POINTER)

Description

Draws an anti-aliased line on the screen. Anti-aliasing is a technique used to smooth the jagged edges that normally appear in lines drawn at an angle.

Parameters

x, *y* - Starting coordinates of line

x2, *y2* -Ending coordinates of line

col - The lines color

buffer - Optional. Can be one of BACKBUFFER (default), FRONTBUFFER or SPRITEBUFFER

Return value

None

Remarks

Anti-aliasing is done through software. Because of this drawing an AA line will be significantly slower than drawing a non anti-aliased line. Horizontal and vertical lines are drawn normally.

See Also: [DrawLine](#), [DrawAlphaLine](#)

Example usage

```
DrawAALine 0,0, 100, 100, RGB(255,255,255)
```

14.12.7 DrawAlphaLine**Syntax**

DrawAlphaLine(*x* as INT,*y* as INT,*x2* as INT,*y2* as INT,*col* as UINT,*alpha* as UINT,*opt buffer* as POINTER)

Description

Draws an alpha blended line. Alpha blending is a technique to make objects partially transparent.

Parameters

x, *y* - Starting coordinates of the line

x2, *y2* - Ending coordinates of the line

col - Color of the line

alpha - Alpha blending value. From 0 to 255

buffer - Optional. Can be one of BACKBUFFER (default), FRONTBUFFER or SPRITEBUFFER

Return value

None

Remarks

Alpha range is from 0 which is invisible to 255 which draws a fully solid line. Alpha blending is a software operation and will be significantly slower than drawing a non blended line.

See Also: [DrawLine](#), [DrawAALine](#)

Example usage

```
DrawAlphaLine 0,0, 640,480, RGB(255,255,255), 128
```

14.12.8 DrawFilledRect**Syntax**

DrawFilledRect(*x* as INT,*y* as INT,*width* as INT,*height* as INT,*col* as UINT,*opt buffer* as POINTER)

Description

Draws a solid colored rectangle.

Parameters

x, *y* - Coordinates of the upper left corner.

width, *height* - Size of the rectangle.

col - Color of the rectangle.

buffer - Optional. Can be one of BACKBUFFER (default), FRONTBUFFER or SPRITEBUFFER.

Return value

None

Remarks

See Also: [DrawRect](#)

Example usage

```
DrawFilledRect 10, 10, 100, 100, RGB(255,0,255)
```

14.12.9 DrawLine

Syntax

DrawLine(x1 as INT,y1 as INT,x2 as INT,y2 as INT,col as UINT,opt buffer as POINTER)

Description

Draws a solid line onto the buffer.

Parameters

x, *y* - Starting coordinates of the line

x2, *y2* - Ending coordinates of the line

col - Color of the line

buffer - Optional. Can be one of BACKBUFFER (default), FRONTBUFFER or SPRITEBUFFER.

Return value

None

Remarks

See Also: [DrawAALine](#), [DrawAlphaLine](#)

Example usage

```
DrawLine 0,0, 100,100, RGB(0,255,0)
```

14.12.1(DRAWMAP

Syntax

DRAWMAP(map as POINTER,opt buffer as POINTER)

Description

Renders the scrolling tile map into the buffer

Parameters

map - Map handle successfully returned by the NEWMAP function

buffer - Optional. Can be one of BACKBUFFER (default), FRONTBUFFER or

SPRITEBUFFER.

Return value

None

Remarks

Maps should be drawn before any other screen element. If rendered into the backbuffer(default) then FLIP must be used to show the changes.

See Also: [NEWMAP](#)

Example usage

```
DRAWMAP myMap
```

14.12.1 DrawRect

Syntax

DrawRect(x as INT,y as INT,width as INT,height as INT,col as UINT,opt buffer as POINTER)

Description

Draws a rectangle onto the buffer. Only the outline of the rectangle is drawn.

Parameters

x, y - Coordinates of the upper left corner.

width, height - Size of the rectangle.

col - Color of the rectangle.

buffer - Optional. Can be one of BACKBUFFER (default), FRONTBUFFER or SPRITEBUFFER.

Return value

None

Remarks

See Also: [DrawFilledRect](#)

Example usage

```
DrawRect 10, 10, 100, 150, RGB(25,100,100)
```

14.12.1 DrawSprite

Syntax

DrawSprite(sprite as POINTER,OPT buffer as POINTER)

Description

Renders the sprite into the buffer.

Parameters

sprite - Sprite pointer returned by LoadSprite
buffer - Optional. Can be one of BACKBUFFER (default), FRONTBUFFER or SPRITEBUFFER.

Return value

None

Remarks

The position of the sprite is set by MoveSprite. To specify the position to render the sprite without changing the internally stored position use DrawSpriteXY

See Also: [MOVESPRITE](#), [DRAWSPRITEXY](#), [LOADSPRITE](#)

Example usage

```
MoveSprite badguy, 100,30
DrawSprite badguy
FLIP
```

14.12.13 DrawSpriteXY

Syntax

DrawSpriteXY(*sprite* as POINTER,*x* as INT,*y* as INT,*OPT buffer* as POINTER)

Description

.Renders a sprite into a buffer at the specified coordinates

Parameters

sprite - Sprite pointer returned by LoadSprite
x, y - Coordinate to render sprite
buffer - Optional. Can be one of BACKBUFFER (default), FRONTBUFFER or SPRITEBUFFER.

Return value

None

Remarks

Does not update the internally stored sprite position. Use when the same sprite is drawn more than once on a screen.

See Also: [DRAWSPRITE](#), [LOADSPRITE](#)

Example usage

```
DrawSpriteXY goodguy, 100, 112
```

14.12.14 FADEPALETTE

Syntax

INT = FADEPALETTE(toCol as UINT,time as UINT)

Description

Automatically Fades an 8 bit screen to the specified RGB color.

Parameters

toCol - RGB color to fade screen to.

time - Total time for fade operation to take.

Return value

None

Remarks

This function will only work with an 8 bit (palletized) screen mode. The time value is specified in 2/25th of a second (0.08). So to specify 8 seconds the value would be $8/0.08 = 100$. Some common values are:

13 = about 1 second

25 = 2 seconds

50 = 4 seconds

100 = 8 seconds

Example usage

CREATESCREEN 640,480,8
SETPALETTECOLOR 2,RGB(255,0,0)
DO
FILLSCREEN 2
WRITETEXT 0,0,"Press ESC to fade and exit"
FLIP
UNTIL KEYDOWN(1)
'4 second FADE to black. 0 = RGB(0,0,0).
FADEPALETTE 0,50
CLOSESCREEN
END

14.12.1 FILLPALETTE

Syntax

INT = FILLPALETTE(col as UINT)

Description

Fills all 255 entries of an 8 bit screen palette with the color specified.

Parameters

col - RGB color to fill palette with.

Return value

None

Remarks

This function only works with 8 bit palletized screen modes. It has no effect on true color screens.

Example usage

```
CREATESCREEN 640,480,8  
FILLPALETTE RGB(100,100,100)
```

14.12.1 FILLSCREEN**Syntax**

FILLSCREEN(*nColor* as UINT, *OPT buffer* as POINTER)

Description

Fills the buffer specified with the RGB color or palette index specified.

Parameters

nColor - Color to use for the fill.

buffer - Optional. Can be one of BACKBUFFER (default), FRONTBUFFER or SPRITEBUFFER

Return value

None

Remarks

For true color screens (16, 24 and 32 bit) the *nColor* parameter is and RGB color. For 8 bit screens the *nColor* parameter is the index of a color in the palette.

Example usage

```
FILLSCREEN RGB(255,0,255)
```

14.12.1 FLIP**Syntax**

INT = FLIP(*OPT bFast*=FALSE as INT)

Description

Flips the backbuffer to the frontbuffer to show drawing changes.

Parameters

bFast - Specifies a VSYNC (FALSE) or fast flip (TRUE).

Return value

Returns the current FPS (Frames Per Second) of the flip. The FPS is calculated as your program calls FLIP and can be used for controlling the timing of your game or program.

Remarks

The *bFast* parameter can be TRUE to flip without waiting for the vertical sync signal or FALSE (default) to specify a normal flipping operation. If FALSE then the maximum FPS your program can achieve is the refresh rate of the monitor. If TRUE then your program will flip as fast as the code will allow but may result in image tearing on some video cards.

In windowed mode the FLIP command uses a blitter operation to copy the contents of the backbuffer to the primary display surface. In a full screen program the back buffer is swapped with the front buffer for an instantaneous update.

Example usage

```

CREATESCREEN 640,480,32
speed = 0
DO
  FILLSCREEN RGB(0,0,255)
  WRITETEXT 0,0,"FPS =" + STR$(speed) + " : Press ESC to close"
  speed = FLIP( TRUE )
UNTIL KEYDOWN(1)
CLOSESCREEN
END

```

14.12.1 FLUSHKEYS

Syntax

FLUSHKEYS

Description

Flushes the DirectInput keyboard buffer and internal key buffer.

Parameters

None

Return value

None

Remarks

Use before calling WAITKEY if you want to be sure no keys have been pressed before hand.

See Also: [WAITKEY](#)

Example usage

```

FLUSHKEYS
WAITKEY

```

14.12.1 FREEMAP

Syntax

FREEMAP(map as POINTER)

Description

Frees both the tile images and map tile position data contained in a map then deletes the map.

Parameters

map - A handle to a map returned by the NEWMAP function

Return value

None

Remarks

The map handle is invalid after this call and must be reinitialized with the NEWMAP function before being used again.

See Also: [NEWMAP](#)

Example usage

```
FREEMAP myMap
```

14.12.2 FreeSprite

Syntax

FreeSprite(sprite as POINTER)

Description

Deletes the sprite specified including all image frames.

Parameters

sprite - A sprite pointer returned by the LoadSprite function

Return value

None

Remarks

The sprite pointer is invalid after a this function is used. It must be reloaded with LoadSprite if you wish to reuse it. All sprites must be freed before the screen is called or memory loss will occur.

See Also: [LOADSPRITE](#)

Example usage

```
FreeSprite badguy
```

14.12.2 GetBufferHeight

Syntax

INT = GetBufferHeight(buffer as POINTER)

Description

Returns the height of the buffer in pixels.

Parameters

buffer - Can be one of FrontBuffer, BackBuffer or SpriteBuffer

Return value

The height of the buffer in pixels.

Remarks

This is the height specified when creating the buffer.

See Also: [GetBufferPitch](#), [GetBufferWidth](#)

Example usage

```
height = GetBufferHeight(BackBuffer)
```

14.12.2: GetBufferPitch**Syntax**

```
UINT = GetBufferPitch(OPT buffer as POINTER)
```

Description

Returns the pitch, or bytes per line, of the buffer specified.

Parameters

buffer - Optional. Can be either BACKBUFFER (default) or FRONTBUFFER

Return value

The pitch of the buffer

Remarks

Used for direct memory access of a screen buffer. The buffer *must* be locked first with the LockBuffer command and unlocked with the UnlockBuffer command after access is completed. You must be sure never to write outside the confines of the buffer.

See Also: [LockBuffer](#), [UnlockBuffer](#), [GetBufferPointer](#)

Example usage

```
DEF buffer as POINTER
IF (CREATESCREEN(800,600,32) < 0)
  MESSAGEBOX 0,"Error creating screen","Error"
ENDIF

DO
  FillScreen 0
  ' turn the pixel at 100,200 blue
  LockBuffer
  buffer = GetBufferPointer
  pitch = GetBufferPitch
  buffer += (200 * pitch) + (100 * 4)
  #<UINT>buffer = RGBToScreen(RGB(0,0,255))
  Unlockbuffer
```

WriteText 0,0,"Press Left MouseButton to Close"
FLIP
UNTIL GetKeyState(0x01)
CLOSESCREEN
END

14.12.2: GetBufferPointer

Syntax

POINTER = GetBufferPointer(opt buffer as POINTER)

Description

Returns the beginning memory address of a screen buffer.

Parameters

buffer - Optional. Can be either BACKBUFFER (default) or FRONTBUFFER

Return value

The address of the first pixel in a screen buffer

Remarks

Used for direct memory access of a screen buffer. The buffer *must* be locked first with the LockBuffer command and unlocked with the UnlockBuffer command after access is completed. You must be sure never to write outside the confines of the buffer.

The return value is actually a UINT. To use with a POINTER a variable must be defined first.

See Also: [LockBuffer](#), [UnlockBuffer](#), [GetBufferPitch](#)

Example usage

DEF buffer as POINTER
IF (CREATESCREEN(800,600,32) < 0)
MESSAGEBOX 0,"Error creating screen","Error"
ENDIF
DO
FillScreen 0
' turn the pixel at 100,200 blue
LockBuffer
buffer = GetBufferPointer
pitch = GetBufferPitch
buffer += (200 * pitch) + (100 * 4)
#<UINT>buffer = RGBToScreen(RGB(0,0,255))
Unlockbuffer
WriteText 0,0,"Press Left MouseButton to Close"
FLIP
UNTIL GetKeyState(0x01)
CLOSESCREEN
END

14.12.2 GetBufferWidth

Syntax

INT = GetBufferWidth(buffer as POINTER)

Description

Returns the width of the buffer in pixels.

Parameters

buffer - Can be one of FrontBuffer, BackBuffer or SpriteBuffer

Return value

The width of the buffer in pixels.

Remarks

This is not the buffers true width, or pitch, but the width specified when creating the buffer.

See Also: [GetBufferPitch](#), [GetBufferHeight](#)

Example usage

```
width = GetBufferWidth(SpriteBuffer)
```

14.12.2 GETJOYSTICKAXISCOUNT

Syntax

INT = GETJOYSTICKAXISCOUNT(OPT device as INT)

Description

Returns the number of axes available on a joystick device.

Parameters

device - Optional device number. Default is 0 or the first joystick attached to the system

Return value

The number of axes.

Remarks

A screen must be open or DirectInput initialized manually before using this function.

Example usage

```
AxisCount = GETJOYSTICKAXISCOUNT(1)
```

14.12.2 GETJOYSTICKBUTTONCOUNT

Syntax

INT = GETJOYSTICKBUTTONCOUNT(opt device as INT)

Description

Returns the number of buttons available on a joystick device.

Parameters

device - Optional device number. Default is 0 or the first joystick attached to the system

Return value

The number of buttons

Remarks

A screen must be open or DirectInput initialized manually before using this function.

Example usage

```
ButtonCount = GETJOYSTICKBUTTONCOUNT(1)
```

14.12.2 GETJOYSTICKCOUNT**Syntax**

```
INT = GETJOYSTICKCOUNT
```

Description

Returns the number of joystick devices attached to the system

Parameters

None

Return value

The number of joystick devices or 0 if none are attached

Remarks

A screen must be open or DirectInput initialized manually before using this function..

Example usage

```
FOR x = 1 TO GETJOYSTICKCOUNT
  PRINT GETJOYSTICKNAME(x-1)
NEXT x
```

14.12.2 GETJOYSTICKNAME**Syntax**

```
STRING = GETJOYSTICKNAME(opt device as INT)
```

Description

Returns the name of a joystick device as shown in the control panel.

Parameters

device - Optional device number. Default is 0 or the first joystick attached to the system

Return value

A string containing the name of the joystick device.

Remarks

A screen must be open or DirectInput initialized manually before using this function.

Example usage

```
FOR x = 1 TO GETJOYSTICKCOUNT
  PRINT GETJOYSTICKNAME(x-1)
NEXT x
```

14.12.2 GETJOYSTICKTYPE**Syntax**

INT = GETJOYSTICKTYPE(opt device as INT)

Description

Returns the type of a joystick device

Parameters

device - Optional device number. Default is 0 or the first joystick attached to the system

Return value

One of the following constants:

@JOYTYPE_UNKNOWN
@JOYTYPE_TRADITIONAL
@JOYTYPE_FLIGHTSTICK
@JOYTYPE_GAMEPAD
@JOYTYPE_RUDDER
@JOYTYPE_WHEEL
@JOYTYPE_HEADTRACKER

Remarks

A screen must be open or DirectInput initialized manually before using this function.

Example usage

```
IF GETJOYSTICKTYPE(0) = @JOYTYPE_GAMEPAD
  PRINT "Its a gamepad!"
ENDIF
```

14.12.3 GETKEY**Syntax**

key = GETKEY

Description

A DirectInput version of the INKEY\$ function. Returns the ascii value of the currently pressed key or an empty string if no key is currently being pressed.

Parameters

None

Return value

The currently pressed key.

Remarks

A screen must be open or DirectInput initialized manually before using this function. GETKEY is non-blocking so it is suitable for use in a game loop. For raw keyboard data such as the arrow and function keys use the KEYDOWN function instead.

See Also: [KEYDOWN](#)

Example usage

```
a$ = GETKEY
IF a$ <> "" THEN b$ = a$
WRITETEXT 0,20,"You Pressed: " + b$
```

14.12.3 GETMAPCOUNT

Syntax

INT = GETMAPCOUNT(map as POINTER)

Description

Returns the the total size, in tiles, of the map

Parameters

map - A map pointer created with the NEWMAP function

Return value

The size of the map in tiles. Equivalent to multiplying GETMAPWIDTH * GETMAPHEIGHT.

Remarks

Used internally by map functions.

Example usage

```
totalsize = GETMAPCOUNT(mymap)
```

14.12.3 GETMAPDATA

Syntax

INT = GETMAPDATA(map as POINTER,x as INT,y as INT)

Description

Gets the current tile number at the specified position. Positions are in map coordinates

Parameters

map - A map pointer created with the NEWMAP function

x, y - Coordinates of the tile in question.

Return value

The tile number that is currently displayed at the coordinates specified.

Remarks

The positions are zero based map coordinates. Map coordinates are specified in tiles instead of pixels.

See Also: [SETMAPDATA](#)

Example usage

```
tile = GETMAPDATA(myMap, 1, 20)
SETMAPDATA myMap, 1, 21, tile
```

14.12.3 GETMAPHEIGHT

Syntax

INT = GETMAPHEIGHT(map as POINTER)

Description

Returns the height of the map in tiles.

Parameters

map - A map pointer created with the NEWMAP function

Return value

The height of the map

Remarks

See Also: [GETMAPWIDTH](#)

Example usage

```
height = GETMAPHEIGHT(myMap)
```

14.12.3 GETMAPPIXELHEIGHT

Syntax

INT = GETMAPPIXELHEIGHT(map as POINTER)

Description

Returns the height, in pixels, of the specified map.

Parameters

map - A map pointer returned by the NEWMAP function.

Return value

The total height of the map in pixels.

Remarks

See Also: [GETMAPPIXELWIDTH](#)

Example usage

```
height = GETMAPPIXELHEIGHT(myMap)
```

14.12.3 GETMAPPIXELWIDTH**Syntax**

INT = GETMAPPIXELWIDTH(map as POINTER)

Description

Returns the width, in pixels, of the specified map.

Parameters

map - A map pointer returned by the NEWMAP function.

Return value

The total width of the map in pixels

Remarks

See Also: [GETMAPPIXELHEIGHT](#)

Example usage

```
width = GETMAPPIXELWIDTH(myMap)
```

14.12.3 GETMAPWIDTH**Syntax**

INT = GETMAPWIDTH(map as POINTER)

Description

Returns the width of a map in tiles.

Parameters

map - A map pointer returned by the NEWMAP function.

Return value

The width of the map

Remarks

See Also: [GETMAPHEIGHT](#)

Example usage

```
width = GETMAPWIDTH(myMap)
```

14.12.3 GETPALETTECOLOR

Syntax

UINT = GETPALETTECOLOR(index as INT)

Description

Returns the RGB color of the specified palette index.

Parameters

index - Index of the color to retrieve.

Return value

The RGB color stored in the palette at the specified index.

Remarks

Index is a number from 0 to 255. This function is only valid for 8 bit screens.

See Also: [SETPALETTECOLOR](#)

Example usage

```
col = GETPALETTECOLOR(0)
```

14.12.3 GetSpriteDelay

Syntax

INT = GetSpriteDelay(sprite as POINTER)

Description

Returns a user defined integer value stored in a sprite.

Parameters

sprite - Sprite pointer returned by the LoadSprite function.

Return value

An integer value

Remarks

The user defined data stored in sprites can be used for any purpose you wish. They are not used by the sprites in any manner.

See Also: [SetSpriteDelay](#)

Example usage

```
delay = GetSpriteDelay(badguy)
```

14.12.3 GetSpriteFrames

Syntax

```
UINT = GetSpriteFrames(sprite as POINTER)
```

Description

Returns the number of image frames available in a sprite.

Parameters

sprite - Sprite pointer returned by the LoadSprite function.

Return value

The total number of frames.

Remarks

Sprites can have any number of individual image frames. Set the currently displayed frame with the SPRITEFRAME command.

See Also: [SPRITEFRAME](#), [LOADSPRITE](#)

Example usage

```
total frames = GetSpriteFrames(goodguy)
IF current frame < total frames
    current frame++
    SPRITEFRAME goodguy, current frame
ELSE
    current frame = 0
ENDIF
```

14.12.4 GetSpriteHeight

Syntax

```
UINT = GetSpriteHeight(sprite as POINTER)
```

Description

Returns the height of the sprite in pixels.

Parameters

sprite - Sprite pointer returned by the LoadSprite function.

Return value

The height of the sprite.

Remarks

LoadSprite can automatically calculate the width and height of a sprite based on the image data. Use this function to retrieve the current height of the sprite.

See Also: [GETSPRITEWIDTH](#)

Example usage

```
sprite = LoadSprite(GETSTARTPATH+"mouth.bmp",0,0,3)
height = GetSpriteHeight(sprite)
```

14.12.4 GetSpritePitch**Syntax**

UINT = GetSpritePitch(sprite as POINTER)

Description

Returns the sprites pitch, or bytes per line, of the internal surface used by the sprites image.

Parameters

sprite - Sprite pointer returned by LoadSprite.

Return value

The pitch of the sprite.

Remarks

Used for direct memory access of the sprites buffer. Sprite must be locked first with LockSprite before using this command.

See Also: [LOADSPRITE](#), [LOCKSPRITE](#), [UNLOCKSPRITE](#), [GETSPRITEPOINTER](#)

Example usage

```
DEF pBuffer as POINTER
sprite1 = LOADSPRITE(GETSTARTPATH+"bug.bmp",0,0,3)
LOCKSPRITE sprite1
pitch = GETSPRITEPITCH sprite1
pBuffer = GETSPRITEPOINTER sprite1
'Change the first pixel of the sprite to red
#<UINT>pBuffer = RGBToScreen(RGB(255,0,0))
'Change the first pixel on the next line to green
pBuffer += pitch
#<UINT>pBuffer = RGBToScreen(RGB(0,255,0))
UNLOCKSPRITE sprite1
```

14.12.4 GetSpritePointer**Syntax**

UINT = GetSpritePointer(sprite as POINTER)

Description

Returns a pointer to the first memory location used by the sprites image. The image is stored on a DirectX surface buffer.

Parameters

sprite - Sprite pointer returned by LoadSprite.

Return value

The address of the first pixel of the sprite

Remarks

Used for direct memory access of a sprites buffer. The buffer *must* be locked first with the LockSprite command and unlocked with the UnlockSprite command after access is completed. You must be sure never to write outside the confines of the buffer, that is the dimensions of the sprite image.

The return value is actually a UINT. To use with a POINTER a variable must be defined first.

See Also: [LOADSPRITE](#), [LOCKSPRITE](#), [GETSPRITEPITCH](#), [UNLOCKSPRITE](#)

Example usage

```
DEF pBuffer as POINTER
sprite1 = LOADSPRITE(GETSTARTPATH+"bug.bmp", 0, 0, 3)
LOCKSPRITE sprite1
pitch = GETSPRITEPITCH sprite1
pBuffer = GETSPRITEPOINTER sprite1
'Change the first pixel of the sprite to red
#<UINT>pBuffer = RGBToScreen(RGB(255,0,0))
'Change the first pixel on the next line to green
pBuffer += pitch
#<UINT>pBuffer = RGBToScreen(RGB(0,255,0))
UNLOCKSPRITE sprite1
```

14.12.4: GetSpriteState**Syntax**

INT = GetSpriteState(sprite as pointer)

Description

Returns a user defined integer value stored in a sprite.

Parameters

sprite - Sprite pointer returned by the LoadSprite function.

Return value

An integer value

Remarks

The user defined data stored in sprites can be used for any purpose you wish. They are not used

by the sprites in any manner.

See Also: [SetSpriteState](#)

Example usage

```
state = GetSpriteState(badguy)
```

14.12.4!GetSpriteType

Syntax

INT = GetSpriteType(sprite as pointer)

Description

Returns a user defined integer value stored in a sprite.

Parameters

sprite - Sprite pointer returned by the LoadSprite function.

Return value

An integer value

Remarks

The user defined data stored in sprites can be used for any purpose you wish. They are not used by the sprites in any manner.

See Also: [SetSpriteType](#)

Example usage

```
type = GetSpriteType(badguy)
```

14.12.4!GetSpriteVelX

Syntax

INT = GetSpriteVelX(sprite as pointer)

Description

Returns a user defined integer value stored in a sprite.

Parameters

sprite - Sprite pointer returned by the LoadSprite function.

Return value

An integer value

Remarks

The user defined data stored in sprites can be used for any purpose you wish. They are not used by the sprites in any manner.

See Also: [SetSpriteVelX](#)

Example usage

```
velocity = GetSpriteVelX(badguy)
```

14.12.4: GetSpriteVelY**Syntax**

INT = GetSpriteVelY(sprite as pointer)

Description

Returns a user defined integer value stored in a sprite.

Parameters

sprite - Sprite pointer returned by the LoadSprite function.

Return value

An integer value

Remarks

The user defined data stored in sprites can be used for any purpose you wish. They are not used by the sprites in any manner.

See Also: [SetSpriteVelY](#)

Example usage

```
velocity = GetSpriteVelY(badguy)
```

14.12.4: GetSpriteWidth**Syntax**

UINT = GetSpriteWidth(sprite as POINTER)

Description

Returns the width of the sprite in pixels.

Parameters

sprite - Sprite pointer returned by the LoadSprite function.

Return value

The width of the sprite.

Remarks

LoadSprite can automatically calculate the width and height of a sprite based on the image data. Use this function to retrieve the current width of the sprite. The width returned is one frame width, not the width of the original image unless the image only contains one frame.

See Also: [GETSPRITEHEIGHT](#)

Example usage

```
sprite = LoadSprite(GETSTARTPATH+"mouth.bmp",0,0,3)
width = GetSpriteWidth(sprite)
```

14.12.4 JOYDOWN

Syntax

INT = JOYDOWN(button as INT, OPT device as INT)

Description

Tests a button on a joystick device.

Parameters

button - Zero based button number to test.

device - Optional device number. Default is 0 or the first joystick attached to the system

Return value

TRUE if the button is down, FALSE otherwise.

Remarks

A screen must be open or DirectInput initialized manually before using this function. This function is non-blocking and is suitable for use in game loops.

Example usage

```
IF JOYDOWN(1) THEN fire missile = TRUE
```

14.12.4 JOYX

Syntax

INT = JOYX(OPT device as INT)

Description

Returns the current X axis position of a joystick device.

Parameters

device - Optional device number. Default is 0 or the first joystick attached to the system.

Return value

The current position of the X axis.

Remarks

A screen must be open or DirectInput initialized manually before using this function. The range of an axis is set by the SETJOYSTICKRANGE function.

See Also: [JOYY](#), [JOYZ](#), [SETJOYSTICKRANGE](#)

Example usage

```
pos = JOYX
```

14.12.5 JOYY**Syntax**

INT = JOYY(opt device=0 as INT)

Description

Returns the current Y axis position of a joystick device.

Parameters

device - Optional device number. Default is 0 or the first joystick attached to the system.

Return value

The current position of the Y axis.

Remarks

A screen must be open or DirectInput initialized manually before using this function. The range of an axis is set by the SETJOYSTICKRANGE function.

See Also: [JOYX](#), [JOYZ](#), [SETJOYSTICKRANGE](#)

Example usage

```
pos = JOYY
```

14.12.5 JOYZ**Syntax**

INT = JOYZ(opt device=0 as INT)

Description

Returns the current Z axis position of a joystick device.

Parameters

device - Optional device number. Default is 0 or the first joystick attached to the system.

Return value

The current position of the Z axis.

Remarks

A screen must be open or DirectInput initialized manually before using this function. The range of an axis is set by the SETJOYSTICKRANGE function.

See Also: [JOYY](#), [JOYX](#), [SETJOYSTICKRANGE](#)

Example usage

```
pos = JOYZ
```

14.12.5:KEYDOWN

Syntax

INT = KEYDOWN(scancode as INT)

Description

Tests for a keypress.

Parameters

scancode - A DirectInput raw scancode to test.

Return value

TRUE if key is currently held down, FALSE if the key is up.

Remarks

A screen must be open or DirectInput initialized manually before using this function. The DirectInput scan codes are listed in the appendix and differ from their Windows virtual key counterparts. This function is non buffered and will indicate the true up/down state of a key at the time of the call. For ASCII input see the GETKEY function.

See Also: [GETKEY](#), [Appendix A](#)

Example usage

```
IF KEYDOWN(1)
```

```
    CLOSESCREEN
```

```
END
```

```
ENDIF
```

14.12.5:LOADMAPDATA

Syntax

INT = LOADMAPDATA(map as POINTER,filename as string,OPT scrollwrap as INT)

Description

Loads tile position data into the map

Parameters

map - Map pointer returned by the NEWMAP function

filename - Name of the file containing the map data.

scrollwrap - Specifies whether the map should wrap around when scrolling.

Return value

TRUE if the map data was successfully loaded and initialized, FALSE if the file could not be opened.

Remarks

See Also: [SAVEMAPDATA](#)

Example usage

```
LOADMAPDATA( myMap, GETSTARTPATH+"level1.dat", TRUE )
```

14.12.5!LOADPALETTE

Syntax

INT = LOADPALETTE(filename as STRING)

Description

Retrieves the palette in an 8 bit bitmap and stores it into the 8 bit screens palette.

Parameters

filename - Name of bitmap file with palette to load

Return value

0 on success or -1 if the palette could not be loaded.

Remarks

The bitmap itself is not loaded, only the palette contained within. This function will only work with an 8 bit (256 color) bitmap and an 8 bit screen. In order for 256 color bitmaps to display correctly the screen must be using the some colors as the bitmap. Use this function to realize the palette of a bitmap.

See Also: [SETPALETTECOLOR](#)

Example usage

```
LOADPALETTE GETSTARTPATH+"pal.bmp"
```

14.12.5!LoadSprite

Syntax

POINTER = LoadSprite(filename as STRING, OPT width as INT, OPT height as INT, OPT frames as INT, OPT bSystemRam as INT)

Description

Loads a sprite from an image file or program resources.

Parameters

filename - File name or resource ID to load sprite from.

width - Optional width of sprite

height - Optional height of sprite

frames - Optional number of frames the image contains

bSystemRam - Optional. If TRUE then sprite is loaded into system memory instead of video memory.

Return value

A pointer to the loaded sprite.

Remarks

LOADSPRITE can handle bitmap, JPEG, and any other image type displayable in Internet Explorer. A screen must be opened before any sprites can be loaded and you must free all sprites with the FREESPRITE function before you close the screen.

If your image consists of a row of frames then you can just specify the number of frames and LoadSprite will calculate the *width* and *height*.

If you know the width and height but the number of frames may vary you can specify just the width and height and the number of frames will be calculated automatically. If your image consists of many rows and columns of frame images you must specify width, height and the number of frames.

The optional *bSystemRam* parameter allows the sprite to be loaded into main system memory instead of video card memory. For alpha blending and rotozooming drawing modes this will result in a dramatic speed increase when displaying the sprite. All other drawing modes should use the default. Loading in system ram is also beneficial if you plan to make many copies of the sprite.

Images can be stored in your programs resources as a scalable image. The resource ID is a string identifier.

See Also: [FREESPRITE](#)

Example usage

```
MySprite = LOADSPRITE(GETSTARTPATH + "badguy.jpg", 0, 0, 10, TRUE)
goodguy = LOADSPRITE(GETSTARTPATH + "goodguy.jpg")
resource sprite = LOADSPRITE("sprite7", 24, 24, 10)
```

14.12.5 LOCKBUFFER

Syntax

INT = LOCKBUFFER(OPT buffer as POINTER)

Description

Locks a buffer into memory for multiple drawing operations or direct memory access.

Parameters

buffer - Optional. Can be one of BACKBUFFER (default), FRONTBUFFER or SPRITEBUFFER.

Return value

Returns the current lock count if the buffer could be locked or -1 on error. The lock count is incremented by 1 every time LOCKBUFFER is called and decremented by 1 every time UNLOCKBUFFER is called. If the lock count is greater than zero then this function simply returns the lockcount for performance reasons.

Remarks

Locking a buffer makes the video card memory directly accessible by your program for direct buffer manipulation. All drawing operations lock the buffer before performing any tasks on the buffer. Every call to LOCKBUFFER must be matched with a call to UNLOCKBUFFER.

Locking a buffer is a time expensive operation so drawing operations can be sped up by locking the buffer before performing many drawing operations at once and then unlocking the buffer before flipping to the display surface.

A locked buffer cannot be flipped, or blitted to so you cannot draw a sprite or image onto a locked buffer using standard commands. You should not keep a display buffer locked longer than the time it takes for one frame to be rendered or your program will suffer performance degradation.

Functions such as WritePixelFast require locking the buffer ahead of time before use.

Any IW BASIC window drawing commands will not work when a buffer is locked. For example the CIRCLE and ELLIPSE commands. This is because the Windows GDI requires an unlocked buffer to work with and performs a lock internally.

See Also: [UNLOCKBUFFER](#)

Example usage

LOCKBUFFER
FOR y=0 to 599
FOR x=0 to 799
WritePixelFast(x,y,c)
NEXT x
NEXT y
UNLOCKBUFFER

14.12.5 LOCKSPRITE

Syntax

INT = LOCKSPRITE(sprite as POINTER)

Description

Locks a sprites buffer to allow direct memory access to the sprites image data.

Parameters

sprite - Sprite pointer returned by LoadSprite.

Return value

Returns the current lock count if the sprite could be locked or -1 on error. The lock count is incremented by 1 every time LOCKSPRITE is called and decremented by 1 every time UNLOCKSPRITE is called. If the lock count is greater than zero then this function simply returns the lock count for performance reasons.

Remarks

A locked sprite allows direct access to the image memory using standard pointers. Every call to LOCKSPRITE must be paired with a call to UNLOCKSPRITE or sprite drawing will fail.

See Also: [UNLOCKSPRITE](#), [LOADSPRITE](#), [GETSPRITEPITCH](#), [GETSPRITEPOINTER](#)

Example usage

```
DEF pBuffer as POINTER
sprite1 = LOADSPRITE(GETSTARTPATH+"bug.bmp", 0, 0, 3)
LOCKSPRITE sprite1
pitch = GETSPRITEPITCH sprite1
pBuffer = GETSPRITEPOINTER sprite1
'Change the first pixel of the sprite to red
#<UINT>pBuffer = RGBToScreen(RGB(255,0,0))
'Change the first pixel on the next line to green
pBuffer += pitch
#<UINT>pBuffer = RGBToScreen(RGB(0,255,0))
UNLOCKSPRITE sprite1
```

14.12.5 MAPDRAWMODE

Syntax

MAPDRAWMODE(*map* as POINTER, *mode* as UINT)

Description

Specifies the drawing mode of the map.

Parameters

map - Map pointer returned by the NEWMAP function

mode - New drawing mode

Return value

None

Remarks

Maps support transparency (mask colors) in the same way sprites do. Mode can be either @BLOCKCOPY or @TRANS. When the drawing mode is set to @TRANS the color specified by MAPMASKCOLOR is not drawn. This allows multiple map layers to be used creating complex 2D environments.

See Also: [MAPMASKCOLOR](#), [DRAWMAP](#), [NEWMAP](#)

Example usage

```
MAPDRAWMODE mymap, @TRANS
```

```
MAPMASKCOLOR mymap, RGB(255,0,0)
```

14.12.5(MAPMASKCOLOR

Syntax

MAPMASKCOLOR(map as POINTER,col as UINT)

Description

Sets the transparency (mask) color to be used when rendering the map onto the buffer.

Parameters

map - Map pointer returned by the NEWMAP function
col - Mask color.

Return value

None

Remarks

Only used when the maps drawing mode is set to @TRANS.

See Also: [NEWMAP](#), [MAPDRAWMODE](#), [DRAWMAP](#)

Example usage

```
MAPDRAWMODE mymap, @TRANS  
MAPMASKCOLOR mymap, RGB(255,0,0)
```

14.12.6(MOUSEDOWN

Syntax

INT = MOUSEDOWN(button as INT)

Description

Returns the current state of a mouse button.

Parameters

button - 1 for left, 2 for right, or 3 for middle button.

Return value

TRUE if button is down, FALSE otherwise.

Remarks

Does not require DirectInput or a screen to be opened. If used with a regular window you must have a normal message handler.

The function returns the state of the physical hardware and does not take into account any button swapping done in the control panel.

Example usage

```
IF MOUSEDOWN(2)
    'right button was pressed
ENDIF
```

14.12.6:MOUSEX**Syntax**

INT = MOUSEX

Description

Returns the current mouse X position in screen coordinates.

Parameters

None.

Return value

The current mouse X position.

Remarks

The mouse position is updated every time your program uses FLIP or waits for a message. This function is not 2D specific and can be used in any windows target.

See Also: [MOUSEY](#)

Example usage

```
xPos = MOUSEX
yPos = MOUSEY
```

14.12.6:MOUSEY**Syntax**

INT = MOUSEY

Description

Returns the current mouse Y position in screen coordinates.

Parameters

None.

Return value

The current mouse Y position.

Remarks

The mouse position is updated every time your program uses FLIP or waits for a message. This function is not 2D specific and can be used in any windows target.

See Also: [MOUSEX](#)

Example usage

```
xPos = MOUSEX  
yPos = MOUSEY
```

14.12.6:MOVEMAP**Syntax**

MOVEMAP(map as POINTER,x as INT,y as INT)

Description

Sets the maps drawing position to the coordinates specified.

Parameters

map - Map pointer returned by the NEWMAP function.

x, y - New position of the upper left corner of the map. In pixels.

Return value

None

Remarks

The coordinates can be negative offsets.

See Also: [SCROLLMAP](#), [NEWMAP](#)

Example usage

```
MOVEMAP mymap, -10, 0
```

14.12.6:MoveSprite**Syntax**

MoveSprite(sprite as POINTER,x as INT,y as INT)

Description

Updates the sprites position on the screen.

Parameters

sprite - Sprite pointer returned by the LoadSprite function

x, y - New position for the sprite

Return value

None

Remarks

The sprites position is used by the DRAWSprite command.

See Also: [DRAWSPRITE](#), [DRAWSPRITEXY](#), [LOADSPRITE](#)

Example usage

```
MOVESPRITE badguy, 100, 17
DRAWSPRITE badguy
FLIP
```

14.12.6 NEWMAP

Syntax

POINTER = NEWMAP(filename as STRING,opt width=0 as INT,opt height=0 as INT,opt frames=0 as INT)

Description

Creates a scrolling tile map and loads the tile images.

Parameters

filename - File name or resource ID to load tile images from.

width - Optional. width of one tile

height - Optional height of one tile

frames - Optional number of tiles the image contains

Return value

A pointer to the newly created map or NULL if the tile images could not be loaded.

Remarks

NEWMAP uses the same image loading routine as LoadSprite. After the map is created and the tile images loaded you must describe where the tiles are placed in the map with the LOADMAPDATA or CREATEMAPDATA/SETMAPDATA functions.

See Also: [LOADSPRITE](#), [LOADMAPDATA](#), [CREATEMAPDATA](#), [SETMAPDATA](#)

Example usage

```
map = NEWMAP(GETSTARTPATH+"mapdata.bmp", 64, 64)
IF map = NULL
  CLOSESCREEN
  MESSAGEBOX 0,"failed to load map images","error"
END
ENDIF
IF LOADMAPDATA(map,GETSTARTPATH+"Level.dat") = FALSE
  CLOSESCREEN
  MESSAGEBOX 0,"failed to load map data","error"
END
ENDIF
```

14.12.6 PALETTEINDEX

Syntax

UINT = PALETTEINDEX(index as INT)

Description

Used to allow GDI drawing functions to use the palette of an 8 bit screen.

Parameters

index - Palette index to retrieve color from.

Return value

A special palette constant and color.

Remarks

IWBASIC primitive drawing commands such as CIRCLE and ELLIPSE expect an RGB color to be specified. In order to properly use these commands with a DirectX 8 bit screen a special color constant must be specified. If an RGB color is passed directly to the primitive drawing commands a best guess match from the palette will be used. Note that this command is not necessary for any of the 2D drawing commands.

Example usage

```
CIRCLE BackBuffer,110,125,50,PALETTEINDEX(3),PALETTEINDEX(4)
```

14.12.6 ReadPixel

Syntax

UINT = ReadPixel(x as INT,y as INT,opt buffer as POINTER)

Description

Returns the RGB color or palette index of a pixel at the specified coordinates.

Parameters

x, *y* - coordinates of the pixel to read.

buffer - Optional. Can be either BACKBUFFER (default) or FRONTBUFFER.

Return value

The RGB color, or palette index, of the pixel.

Remarks

Because of color conversions necessary with 16 and 24 bit screens the value read may be different then the value written. The screen automatically converts 32 bit RGB color to the proper pixel format and resolution of the red, green and blue components are adjusted accordingly.

For 8 bit screens the value returned is the palette index of the color.

See Also: [WRITEPIXEL](#), [WRITEPIXELFAST](#)

Example usage

```
col = READPIXEL(100,10)
```

14.12.6:RGBToScreen

Syntax

```
UINT = RGBToScreen(col as UINT)
```

Description

Converts an RGB color to the format used by the currently open screen.

Parameters

col - Color to convert

Return value

The color in the correct pixel format to be directly written to memory.

Remarks

Used for direct memory access. When storing a color directly into buffer memory it must be converted to the correct pixel format first. For 8 bit screens this function is not necessary.

See Also: [GETBUFFERPOINTER](#), [GETBUFFERPITCH](#)

Example usage

```
DEF buffer as POINTER
IF (CREATESCREEN(800,600,32) < 0)
    MESSAGEBOX 0,"Error creating screen","Error"
ENDIF
DO
    FillScreen 0
    ' turn the pixel at 100,200 blue
    LockBuffer
    buffer = GetBufferPointer
    pitch = GetBufferPitch
    buffer += (200 * pitch) + (100 * 4)
    #<UINT>buffer = RGBToScreen(RED(0,0,255))
    Unlockbuffer
    WriteText 0,0,"Press Left MouseButton to Close"
    FLIP
UNTIL GetKeyState(0x01)
CLOSESCREEN
END
```

14.12.6:SAVEMAPDATA

Syntax

```
INT = SAVEMAPDATA(map as POINTER,filename as STRING)
```

Description

Saves the map tile data to a disk file.

Parameters

map - Map pointer returned by NEWMAP.

filename - File to save map tile data to.

Return value

TRUE if the file was created and data saved. FALSE otherwise.

Remarks

See Also: [LOADMAPDATA](#), [SETMAPDATA](#)

Example usage

```
SAVEMAPDATA mymap, "level1.map"
```

14.12.7(SCROLLMAP)

Syntax

SCROLLMAP(*map* as POINTER,*direction* as INT,*amount* as INT)

Description

Scrolls the map in the direction specified.

Parameters

map - Map pointer returned by the NEWMAP function.

direction - A directional constant.

amount - How far, in pixels, to scroll the map.

Return value

None

Remarks

This command updates the internal position of the map by scrolling in the direction specified by the distance specified.

The direction argument can be one of:

@SCROLLUP

@SCROLLDOWN

@SCROLLLEFT

@SCROLLRIGHT

If the tile data was created with scroll wrapping set to TRUE then the map will continuously scroll in the direction specified with each call to SCROLLMAP. The map must still be rendered with DRAWMAP to show the changes made while scrolling. The *amount* argument is a positive integer.

See Also: [MOVEMAP](#)

Example usage

```
DRAWMAP map
IF KEYDOWN(DIK UP)
    ScrollMap(map,@SCROLLUP,2)
```

```

ENDIF
IF KEYDOWN(DIK_DOWN)
    ScrollMap(map,@SCROLLDOWN,2)
ENDIF
IF KEYDOWN(DIK_LEFT)
    ScrollMap(map,@SCROLLLEFT,2)
ENDIF
IF KEYDOWN(DIK_RIGHT)
    ScrollMap(map,@SCROLLRIGHT,2)
ENDIF
FLIP

```

14.12.7 SETJOYSTICKDEADZONE

Syntax

SETJOYSTICKDEADZONE(*Axis* as INT,*pct* as FLOAT,*OPT device* as INT)

Description

Sets the deadzone of a joystick axis.

Parameters

Axis - The axis to set.

pct - Deadzone percentage.

device - Optional device number. Default is 0 or the first joystick attached to the system

Return value

None

Remarks

A deadzone is a percentage of the range of movement about the center of the axis where the joystick will report being at the center of its range. The default dead zone is 10% and percentages are entered directly as in 50.5 would be equal to 50.5%. *Axis* is one of the constants @XAXIS, @YAXIS or @ZAXIS.

The deadzone should be set after the range for the axis. A screen must be open or DirectInput manually initialized before using this function.

See Also: [SETJOYSTICKRANGE](#)

Example usage

```
SETJOYSTICKDEADZONE @XAXIS, 15.0
```

14.12.7 SETJOYSTICKRANGE

Syntax

SETJOYSTICKRANGE(*Axis* as INT,*Min* as INT,*Max* as INT,*OPT device* as INT)

Description

Sets the minimum and maximum values an axis will report.

Parameters

Axis - The axis to set.

Min - Minimum range

Max - Maximum range

device - Optional device number. Default is 0 or the first joystick attached to the system

Return value

None

Remarks

Default range is from -1000 to +1000. Values are integer and *Min* must be less than *Max*. *Axis* is one of the constants @XAXIS, @YAXIS or @ZAXIS. A joystick axis normally reports 0 at its midpoint.

See Also: [SETJOYSTICKDEADZONE](#)

Example usage

```
SETJOYSTICKRANGE @YAXIS, -200, 200
```

14.12.7: SETMAPDATA**Syntax**

SETMAPDATA(map as POINTER,x as INT,y as INT,tile as INT)

Description

Sets the tile image number to display at the specified map coordinates.

Parameters

map - Map pointer returned by the NEWMAP function.

x, y - Coordinates to set the tile number.

tile - New tile image number.

Return value

None

Remarks

Each frame (tile) in an image is assigned a number starting from 0. The tile image number specifies which tile will be drawn at the specified coordinate. Coordinates are in map units which are tiles wide by tiles high.

A value less than zero or greater than the number of tiles in the image will cause that map coordinate not to be drawn when rendering with DRAWMAP. This can be used to create patterns of missing tiles and to show the background through the map.

See Also: [GETMAPDATA](#), [DRAWMAP](#)

Example usage

```
SETMAPDATA mymap, 0, 0, 10
```

14.12.7 SETMAPVIEWPORT**Syntax**

SETMAPVIEWPORT(*map* as POINTER, *vpRect* as WINRECT)

Description

Sets the rendering viewport of the map.

Parameters

map - Map pointer returned by the NEWMAP function

vpRect - Rectangle describing the viewport

Return value

None

Remarks

The view port is a rectangular area that the map is rendered into this is set by default to the same size as the screen dimensions. You can restrict the area the map renders into by defining the viewport rectangle. When using a viewport the map automatically shifts the map rendering to match the upper left corner of the viewport as position 0,0.

See Also: [DRAWMAP](#)

Example usage

```
'assuming a 640 x 480 screen size
'dont render the map on the first 40 lines
DEF vp as WINRECT
vp.top = 40
vp.bottom = 480
vp.left = 0
vp.right = 640
SETMAPVIEWPORT map, vp
```

14.12.7 SETPALETTECOLOR**Syntax**

INT = SETPALETTECOLOR(*index* as INT, *col* as UINT)

Description

Sets the color of a palette index in an 8 bit screen.

Parameters

index - The palette index to change.

col - The RGB color to set.

Return value

0 on success or -1 if the palette could not be changed.

Remarks

index is a palette number from 0 to 255. This function only works with 8 bit screen modes and has no effect on true color screens.

See Also: [LOADPALETTE](#), [GETPALETTECOLOR](#)

Example usage

```
SETPALETTECOLOR 2, RGB(255,255,0)
FILLSCREEN 2
```

14.12.7 SetSpriteDelay

Syntax

SetSpriteDelay(sprite as pointer,info as INT)

Description

Stores a user defined integer into a sprite.

Parameters

sprite - Sprite pointer returned by the LoadSprite function.

info - The integer to store

Return value

None

Remarks

The user defined data stored in sprites can be used for any purpose you wish. They are not used by the sprites in any manner.

See Also: [GetSpriteDelay](#)

Example usage

```
SetSpriteDelay badguy, 24
```

14.12.7 SetSpriteState

Syntax

SetSpriteState(sprite as pointer,info as INT)

Description

Stores a user defined integer into a sprite.

Parameters

sprite - Sprite pointer returned by the LoadSprite function.
info - The integer to store

Return value

None

Remarks

The user defined data stored in sprites can be used for any purpose you wish. They are not used by the sprites in any manner.

See Also: [GetSpriteState](#)

Example usage

```
SetSpriteState badguy, 5
```

14.12.7!SetSpriteType

Syntax

SetSpriteType(sprite as pointer,info as INT)

Description

Stores a user defined integer into a sprite.

Parameters

sprite - Sprite pointer returned by the LoadSprite function.
info - The integer to store

Return value

None

Remarks

The user defined data stored in sprites can be used for any purpose you wish. They are not used by the sprites in any manner.

See Also: [GetSpriteType](#)

Example usage

```
SetSpriteType badguy, 24
```

14.12.7!SetSpriteVelX

Syntax

SetSpriteVelX(sprite as pointer,info as INT)

Description

Stores a user defined integer into a sprite.

Parameters

sprite - Sprite pointer returned by the LoadSprite function.

info - The integer to store

Return value

None

Remarks

The user defined data stored in sprites can be used for any purpose you wish. They are not used by the sprites in any manner.

See Also: [GetSpriteVelX](#)

Example usage

```
SetSpriteVelX badguy, 100
```

14.12.8(SetSpriteVelY**Syntax**

SetSpriteVelY(sprite as pointer,info as INT)

Description

Stores a user defined integer into a sprite.

Parameters

sprite - Sprite pointer returned by the LoadSprite function.

info - The integer to store

Return value

None

Remarks

The user defined data stored in sprites can be used for any purpose you wish. They are not used by the sprites in any manner.

See Also: [GetSpriteVelY](#)

Example usage

```
SetSpriteVelY badguy, 50
```

14.12.8'SpriteAlpha**Syntax**

SpriteAlpha(sprite as POINTER,alpha as WORD)

Description

Sets the alpha blending value for drawing with sprite modes @ALPHA, @TRANSALPHA, and @TRANSSHADOW

Parameters

sprite - Sprite pointer returned by LoadSprite.

alpha - The alpha blending value

Return value

None

Remarks

The alpha value ranges from 0 fully solid to 255 fully transparent.

See Also: [SpriteDrawMode](#)

Example usage

```
SpriteAlpha badguy, 128
```

14.12.8:SpriteAngle

Syntax

SpriteAngle(sprite as POINTER,angle as DOUBLE)

Description

Sets the rotation angle in radians for drawing modes @ROTOZOOM and @TRANSROTOZOOM

Parameters

sprite - Sprite pointer returned by LoadSprite.

angle - The rotation angle in radians.

Return value

None

Remarks

To use degrees multiply the degree by .01745. Rotations and scaling are drawing properties only. They do not change the sprite itself, only the way it appears on the screen. Standard hit testing will still use the original image.

See Also: [SpriteScaleFactor](#)

Example usage

```
SpriteAngle ship, 90 * ,01745
```

14.12.8:SpriteCollided

Syntax

UINT = SpriteCollided(sprite1 as POINTER, sprite2 as POINTER, OPT bypixel as INT)

Description

Tests for collisions between two sprites.

Parameters

sprite1 - A sprite pointer.

sprite2 - A sprite pointer.

bypixel - Optional. Specifies pixel perfect mode.

Return value

TRUE if sprites collided or FALSE otherwise.

Remarks

If *bypixel* is TRUE then the sprites are compared including transparency (mask color) information and will return TRUE only if a non-transparent pixel on both sprites are overlapping. If *bypixel* is FALSE then the function will compare the bounding rectangles of the sprites only and return TRUE if any part of the rectangles overlap.

The sprites positions on the screen are determined by the internal coordinates set by the MoveSprite command. The current image frame is determined by the current frame set by the SpriteFrame command. The sprites must refer to two different sprites. If the same sprite pointer is used for both *sprite1* and *sprite2* the function returns FALSE regardless of position. Use SpriteCollidedEx for a position independent collision test that works with a single sprite.

See Also: [MoveSprite](#), [SpriteCollidedEx](#)

Example usage

```
IF SpriteCollided(goodguy, badguy, TRUE)
  EndGame ( )
ENDIF
```

14.12.8 SpriteCollidedEx

Syntax

UINT = SpriteCollidedEx(sprite1 as POINTER, x1 as INT, y1 as INT, frame1 as INT, sprite2 as POINTER, x2 as INT, y2 as INT, frame2 as INT)

Description

Tests for collisions between sprites

Parameters

sprite1 - A sprite pointer

x1, y1 - The position of the first sprite

frame1 - The image frame of the first sprite

sprite2 - A sprite pointer

x2, y2 - The position of the second sprite
frame2 - The image frame of the second sprite

Return value

TRUE if the sprites are colliding, FALSE otherwise.

Remarks

The collision test is always done in pixel perfect mode. The internal position and frame of the sprites is not used or modified by this function. The same sprite pointer can be used for both sprites allowing virtual testing. Image frames for sprites are zero based.

See Also: SpriteCollided

Example usage

```
IF SpriteCollidedEx( goodguy, ggX, ggY, ggF, badguy, bgX, bgY, ggF)
    EndGame()
ENDIF
```

14.12.8!SpriteDrawMode

Syntax

SpriteDrawMode(sprite as POINTER,mode as UINT)

Description

Sets the current drawing mode of the sprite

Parameters

sprite - Sprite pointer returned by LoadSprite
mode - The new drawing mode

Return value

None

Remarks

mode can be one of the following:

- @BLOCKCOPY
- @ALPHA
- @SCALED
- @ROTOZOOM
- @HFLIP
- @VFLIP
- @TRANS
- @TRANSALPHA
- @TRANSSCALED
- @TRANSSHADOW
- @TRANSROTOZOOM

@TRANSHFLIP

@TRANSVFLIP

Example usage

```
SpriteDrawMode goodguy, @TRANS
```

14.12.8 SpriteFrame

Syntax

SpriteFrame(sprite as POINTER, frame as UINT)

Description

Sets the current image frame of the sprite

Parameters

sprite - Sprite pointer returned by LoadSprite.

frame - The zero based image frame.

Return value

None

Remarks

The image frame is sometimes referred to as the animation frame. A sprites image can contain one or more individual frames. The number of frames in an image is determined by the LoadSprite function.

See Also: [LoadSprite](#)

Example usage

```
SpriteFrame badguy, 10
```

14.12.8 SpriteMaskColor

Syntax

SpriteMaskColor(sprite as POINTER, clr as UINT)

Description

Sets the mask color of transparent sprites.

Parameters

sprite - Sprite pointer returned by LoadSprite.

clr - The transparency color or color index to use.

Return value

None

Remarks

The mask color is used by drawing modes beginning with `@TRANS..` It specifies a color in the sprites image that will not be shown when rendering the sprite onto the buffer with `DrawSprite` or `DrawSpriteXY`. The mask color is also used in collision testing to determine what pixels are invisible. For 8 bit screens the masking color is specified as a palette index instead of an RGB color.

See Also: [LoadSprite](#), [DrawSprite](#), [DrawSpriteXY](#), [SpriteDrawMode](#), [SpriteCollided](#), [SpriteCollidedEx](#)

Example usage

```
SpriteMaskColor badguy, RGB(255,255,255)
```

14.12.8 SpriteScaleFactor

Syntax

`SpriteScaleFactor(sprite as POINTER,scale as float)`

Description

Sets the scaling value for the sprite drawing modes `@SCALED`, `@TRANSSCALED`, `@ROTOZOOM` and `@TRANSROTOZOOM`.

Parameters

sprite - Sprite pointer returned by `LoadSprite`.

scale - The scale factor to use.

Return value

None

Remarks

The scaling factor is 1.0 by default. Use 2.0 for a double sized sprite, 0.5 for a half sized, etc. If your just scaling the sprite use the modes `@SCALED` and `@TRANSSCALED` as they are much faster than the `ROTOZOOM` routines which work by software texture mapping.

See Also: [LoadSprite](#), [SpriteAngle](#)

Example usage

```
SpriteScaleFactor goodguy, 2.5
```

14.12.8 SpriteShadowOffset

Syntax

`SpriteShadowOffset(sprite as POINTER,x as INT,y as INT)`

Description

Specifies the offset position to draw a sprites shadow with the drawing mode `@TRANSSHADOW`.

Parameters

sprite - Sprite pointer returned by LoadSprite.

x, y - The offsets to position the shadow.

Return value

None

Remarks

Offsets are in negative directions. The transparency of the shadow is determined by the SpriteAlpha function. Note that with the @TRANSSHADOW drawing mode that only the shadow is drawn. You must switch modes to draw the actual sprite image.

See Also: [SpriteAlpha](#), [LoadSprite](#)

Example usage

SpriteAlpha sprite2,180
SpriteShadowOffset sprite2,-20,-20
...
SpriteDrawMode sprite2,@TRANSSHADOW
DrawSpriteXY sprite2,x2,y2
SpriteDrawMode sprite2,@TRANS
DrawSpriteXY sprite2,x2,y2

14.12.9(SpriteToBuffer**Syntax**

POINTER = SpriteToBuffer(sprite as POINTER)

Description

Attaches the sprite to the SpriteBuffer

Parameters

sprite - Sprite pointer returned by LOADSPRITE or CREATESPRITE

Return value

For convenience a pointer to the SpriteBuffer variable.

Remarks

The SpriteBuffer buffer is a special pseudo DirectX buffer than can be used to draw on a sprite with any of the 2D or Window drawing functions. This command attaches a sprite to that buffer.

See Also: [LOADSPRITE](#), [CREATESPRITE](#)

Example usage

newSprite = CREATESPRITE(100,100,1)
SpriteToBuffer(newSprite)
FILLSCREEN RGB(255,0,0), SpriteBuffer
DrawRect 0,0,100,100,RGB(0,0,255), SpriteBuffer

14.12.9 UNLOCKBUFFER

Syntax

INT = UNLOCKBUFFER(OPT buffer as POINTER)

Description

Unlocks a previously locked buffer.

Parameters

buffer - Optional. Can be either BACKBUFFER (default) or FRONTBUFFER

Return value

Returns the current lock count if the buffer could be locked or -1 on error.

Remarks

Locking a buffer makes the video card memory directly accessible by your program for direct buffer manipulation. All drawing operations lock the buffer before performing any tasks on the buffer. Every call to LOCKBUFFER must be matched with a call to UNLOCKBUFFER.

The buffer must be unlocked before it can be flipped, blitted to, or drawn on with the GDI.

See Also: [LOCKBUFFER](#)

Example usage

LOCKBUFFER
FOR y=0 to 599
FOR x=0 to 799
WritePixelFast(x, y, c)
NEXT x
NEXT y
UNLOCKBUFFER

14.12.9 UNLOCKSPRITE

Syntax

INT = UNLOCKSPRITE(sprite as POINTER)

Description

Unlocks a previously locked sprite.

Parameters

sprite - Sprite pointer returned by LoadSprite.

Return value

The current lock count of the sprite or -1 on error.

Remarks

A locked sprite allows direct access to the image memory using standard pointers. Every call to

LOCKSPRITE must be paired with a call to UNLOCKSPRITE or sprite drawing will fail.

See Also: [LOCKSPRITE](#), [LOADSPRITE](#), [GETSPRITEPITCH](#), [GETSPRITEPOINTER](#)

Example usage

```
DEF pBuffer as POINTER
sprite1 = LOADSPRITE(GETSTARTPATH+"bug.bmp",0,0,3)
LOCKSPRITE sprite1
pitch = GETSPRITEPITCH sprite1
pBuffer = GETSPRITEPOINTER sprite1
'Change the first pixel of the sprite to red
#<UINT>pBuffer = RGBToScreen(RGB(255,0,0))
'Change the first pixel on the next line to green
pBuffer += pitch
#<UINT>pBuffer = RGBToScreen(RGB(0,255,0))
UNLOCKSPRITE sprite1
```

14.12.9.3 WAITKEY

Syntax

WAITKEY(OPT keycode as INT)

Description

Pauses your program and waits for a key to be pressed.

Parameters

keycode - Optional DirectInput scancode to wait for.

Return value

None

Remarks

A screen must be open or DirectInput initialized manually before using this function. The DirectInput scan codes are listed in the appendix and differ from their Windows virtual key counterparts. If keycode is not supplied then the function will wait for any key to be pressed. Otherwise the function waits for the the specified key to be pressed.

See Also: [GETKEY](#), [KEYDOWN](#), [Appendix A](#)

Example usage

```
WRITETEXT 10,100,"Press any key to close"
WAITKEY
CLOSESCREEN
END
```

14.12.9.4 WriteAlphaPixel

Syntax

WriteAlphaPixel(x as INT,y as INT,col as UINT,alpha as INT,opt buffer as POINTER)

Description

Sets one pixel on a buffer to the specified color using alpha blending.

Parameters

x, *y* - Coordinates of pixel to set.

col - RGB color or color index to set pixel.

alpha - Alpha blending value

buffer - Optional. Can be one of BACKBUFFER (default), FRONTBUFFER or SPRITEBUFFER.

Return value

None

Remarks

The alpha blending value ranges from 0 to 255.

See Also: [WritePixel](#), [WritePixelFast](#)

Example usage

```
WriteAlphaPixel 100,100, RGB(155,0,255), 128
```

14.12.9 WritePixel**Syntax**

WritePixel(*x* as INT,*y* as INT,*col* as UINT,*OPT buffer* as POINTER)

Description

Sets one pixel on the buffer to the color or color index specified.

Parameters

x, *y* - Coordinates of pixel to set.

col - RGB color or color index to set pixel.

buffer - Optional. Can be one of BACKBUFFER (default), FRONTBUFFER or SPRITEBUFFER.

Return value

None

Remarks

See Also: [WriteAlphaPixel](#), [WritePixelFast](#)

Example usage

```
WritePixel 100,100, RGB(155,0,255)
```

14.12.9 WritePixelFast**Syntax**

WritePixelFast(*x* as INT,*y* as INT,*col* as UINT,*OPT buffer* as POINTER)

Description

A highly optimized pixel setting function.

Parameters

x, *y* - Coordinates of pixel to set.

col - RGB color or color index to set pixel.

buffer - Optional. Can be one of BACKBUFFER (default), FRONTBUFFER or SPRITEBUFFER.

Return value

None

Remarks

This function REQUIRES a locked buffer to work with. WritePixelFast is a highly optimized hand coded machine language function will give the best performance in games when a large number of pixels need to be changed at once. Boundary checking is not done so you must be careful not to write outside of the confines of the buffer or your program will generate a fault.

See Also: [WritePixel](#), [WriteAlphaPixel](#)

Example usage

LOCKBUFFER
FOR y=0 to 599
FOR x=0 to 799
WritePixelFast(x,y,c)
NEXT x
NEXT y
UNLOCKBUFFER

14.12.9 WriteText

Syntax

WriteText(*x* as int,*y* as int,*str* as STRING, *OPT buffer* as POINTER)

Description

Writes text into a buffer

Parameters

x, *y* - Coordinates of text in pixels.

str - The text to write

buffer - Optional. Can be one of BACKBUFFER (default), FRONTBUFFER or SPRITEBUFFER.

Return value

None

Remarks

WriteText uses the current font and drawing mode to quickly print a line of text into the buffer. The font and drawing mode can be set with the IW BASIC commands SETFONT and DRAWMODE. For time critical application WriteText is significantly faster than using PRINT.

Example usage

```
DRAWMODE BackBuffer,@TRANSPARENT
SETFONT BackBuffer,"Courier New",12,400
WRITETEXT 10,10, "Speed = " + STR$(fps)
```

14.13 Appendix**14.13.1 DirectInput Keyboard Codes**

The DirectInput scan codes listed in IW BASIC source format. Copy and paste any needed directly into your program.

```
/
*****
*****
*
* DirectInput keyboard scan codes
*
*****
*****/
CONST DIK_ESCAPE = 0x01
CONST DIK_1 = 0x02
CONST DIK_2 = 0x03
CONST DIK_3 = 0x04
CONST DIK_4 = 0x05
CONST DIK_5 = 0x06
CONST DIK_6 = 0x07
CONST DIK_7 = 0x08
CONST DIK_8 = 0x09
CONST DIK_9 = 0x0A
CONST DIK_0 = 0x0B
CONST DIK_MINUS = 0x0C /* - on main keyboard */
CONST DIK_EQUALS = 0x0D
CONST DIK_BACK = 0x0E /* backspace */
CONST DIK_TAB = 0x0F
CONST DIK_Q = 0x10
CONST DIK_W = 0x11
CONST DIK_E = 0x12
CONST DIK_R = 0x13
CONST DIK_T = 0x14
CONST DIK_Y = 0x15
```

```
CONST DIK_U = 0x16
CONST DIK_I = 0x17
CONST DIK_O = 0x18
CONST DIK_P = 0x19
CONST DIK_LBRACKET = 0x1A
CONST DIK_RBRACKET = 0x1B
CONST DIK_RETURN = 0x1C /* Enter on main keyboard */
CONST DIK_LCONTROL = 0x1D
CONST DIK_A = 0x1E
CONST DIK_S = 0x1F
CONST DIK_D = 0x20
CONST DIK_F = 0x21
CONST DIK_G = 0x22
CONST DIK_H = 0x23
CONST DIK_J = 0x24
CONST DIK_K = 0x25
CONST DIK_L = 0x26
CONST DIK_SEMICOLON = 0x27
CONST DIK_APOSTROPHE = 0x28
CONST DIK_GRAVE = 0x29 /* accent grave */
CONST DIK_LSHIFT = 0x2A
CONST DIK_BACKSLASH = 0x2B
CONST DIK_Z = 0x2C
CONST DIK_X = 0x2D
CONST DIK_C = 0x2E
CONST DIK_V = 0x2F
CONST DIK_B = 0x30
CONST DIK_N = 0x31
CONST DIK_M = 0x32
CONST DIK_COMMA = 0x33
CONST DIK_PERIOD = 0x34 /* . on main keyboard */
CONST DIK_SLASH = 0x35 /* / on main keyboard */
CONST DIK_RSHIFT = 0x36
CONST DIK_MULTIPLY = 0x37 /* * on numeric keypad */
CONST DIK_LMENU = 0x38 /* left Alt */
CONST DIK_SPACE = 0x39
CONST DIK_CAPITAL = 0x3A
CONST DIK_F1 = 0x3B
CONST DIK_F2 = 0x3C
CONST DIK_F3 = 0x3D
CONST DIK_F4 = 0x3E
CONST DIK_F5 = 0x3F
CONST DIK_F6 = 0x40
CONST DIK_F7 = 0x41
CONST DIK_F8 = 0x42
CONST DIK_F9 = 0x43
```

```
CONST DIK_F10 = 0x44
CONST DIK_NUMLOCK = 0x45
CONST DIK_SCROLL = 0x46 /* Scroll Lock */
CONST DIK_NUMPAD7 = 0x47
CONST DIK_NUMPAD8 = 0x48
CONST DIK_NUMPAD9 = 0x49
CONST DIK_SUBTRACT = 0x4A /* - on numeric keypad */
CONST DIK_NUMPAD4 = 0x4B
CONST DIK_NUMPAD5 = 0x4C
CONST DIK_NUMPAD6 = 0x4D
CONST DIK_ADD = 0x4E /* + on numeric keypad */
CONST DIK_NUMPAD1 = 0x4F
CONST DIK_NUMPAD2 = 0x50
CONST DIK_NUMPAD3 = 0x51
CONST DIK_NUMPAD0 = 0x52
CONST DIK_DECIMAL = 0x53 /* . on numeric keypad */
CONST DIK_OEM_102 = 0x56 /* < > | on UK/Germany keyboards
*/
CONST DIK_F11 = 0x57
CONST DIK_F12 = 0x58
CONST DIK_F13 = 0x64 /* (NEC PC98) */
CONST DIK_F14 = 0x65 /* (NEC PC98) */
CONST DIK_F15 = 0x66 /* (NEC PC98) */
CONST DIK_KANA = 0x70 /* (Japanese keyboard) */
CONST DIK_ABNT_C1 = 0x73 /* / ? on Portugese (Brazilian)
keyboards */
CONST DIK_CONVERT = 0x79 /* (Japanese keyboard) */
CONST DIK_NOCONVERT = 0x7B /* (Japanese keyboard) */
CONST DIK_YEN = 0x7D /* (Japanese keyboard) */
CONST DIK_ABNT_C2 = 0x7E /* Numpad . on Portugese
(Brazilian) keyboards */
CONST DIK_NUMPADEQUALS = 0x8D /* = on numeric keypad (NEC
PC98) */
CONST DIK_PREVTRACK = 0x90 /* Previous Track
(DIK_CIRCUMFLEX on Japanese keyboard) */
CONST DIK_AT = 0x91 /* (NEC PC98) */
CONST DIK_COLON = 0x92 /* (NEC PC98) */
CONST DIK_UNDERLINE = 0x93 /* (NEC PC98) */
CONST DIK_KANJI = 0x94 /* (Japanese keyboard) */
CONST DIK_STOP = 0x95 /* (NEC PC98) */
CONST DIK_AX = 0x96 /* (Japan AX) */
CONST DIK_UNLABELED = 0x97 /* (J3100) */
CONST DIK_NEXTTRACK = 0x99 /* Next Track */
CONST DIK_NUMPADENTER = 0x9C /* Enter on numeric keypad */
CONST DIK_RCONTROL = 0x9D
CONST DIK_MUTE = 0xA0 /* Mute */
```

```

CONST DIK_CALCULATOR = 0xA1 /* Calculator */
CONST DIK_PLAYPAUSE = 0xA2 /* Play / Pause */
CONST DIK_MEDIASTOP = 0xA4 /* Media Stop */
CONST DIK_VOLUMEDOWN = 0xAE /* Volume - */
CONST DIK_VOLUMEUP = 0xB0 /* Volume + */
CONST DIK_WEBHOME = 0xB2 /* Web home */
CONST DIK_NUMPADCOMMA = 0xB3 /* , on numeric keypad (NEC
PC98) */
CONST DIK_DIVIDE = 0xB5 /* / on numeric keypad */
CONST DIK_SYSRQ = 0xB7
CONST DIK_RMENU = 0xB8 /* right Alt */
CONST DIK_PAUSE = 0xC5 /* Pause */
CONST DIK_HOME = 0xC7 /* Home on arrow keypad */
CONST DIK_UP = 0xC8 /* UpArrow on arrow keypad */
CONST DIK_PRIOR = 0xC9 /* PgUp on arrow keypad */
CONST DIK_LEFT = 0xCB /* LeftArrow on arrow keypad */
CONST DIK_RIGHT = 0xCD /* RightArrow on arrow keypad */
CONST DIK_END = 0xCF /* End on arrow keypad */
CONST DIK_DOWN = 0xD0 /* DownArrow on arrow keypad */
CONST DIK_NEXT = 0xD1 /* PgDn on arrow keypad */
CONST DIK_INSERT = 0xD2 /* Insert on arrow keypad */
CONST DIK_DELETE = 0xD3 /* Delete on arrow keypad */
CONST DIK_LWIN = 0xDB /* Left Windows key */
CONST DIK_RWIN = 0xDC /* Right Windows key */
CONST DIK_APPS = 0xDD /* AppMenu key */
CONST DIK_POWER = 0xDE /* System Power */
CONST DIK_SLEEP = 0xDF /* System Sleep */
CONST DIK_WAKE = 0xE3 /* System Wake */
CONST DIK_WEBSEARCH = 0xE5 /* Web Search */
CONST DIK_WEBFAVORITES = 0xE6 /* Web Favorites */
CONST DIK_WEBREFRESH = 0xE7 /* Web Refresh */
CONST DIK_WEBSTOP = 0xE8 /* Web Stop */
CONST DIK_WEBFORWARD = 0xE9 /* Web Forward */
CONST DIK_WEBBACK = 0xEA /* Web Back */
CONST DIK_MYCOMPUTER = 0xEB /* My Computer */
CONST DIK_MAIL = 0xEC /* Mail */
CONST DIK_MEDIASELECT = 0xED /* Media Select */
/*
* Alternate names for keys, to facilitate transition from
DOS.
*/
CONST DIK_BACKSPACE = DIK_BACK /* backspace */
CONST DIK_NUMPADSTAR = DIK_MULTIPLY /* * on numeric keypad
*/
CONST DIK_LALT = DIK_LMENU /* left Alt */
CONST DIK_CAPSLOCK = DIK_CAPITAL /* CapsLock */

```

```
CONST DIK_NUMPADMINUS = DIK_SUBTRACT /* - on numeric keypad
*/
CONST DIK_NUMPADPLUS = DIK_ADD /* + on numeric keypad */
CONST DIK_NUMPADPERIOD = DIK_DECIMAL /* . on numeric keypad
*/
CONST DIK_NUMPADSLASH = DIK_DIVIDE /* / on numeric keypad
*/
CONST DIK_RALT = DIK_RMENU /* right Alt */
CONST DIK_UPARROW = DIK_UP /* UpArrow on arrow keypad */
CONST DIK_PGUP = DIK_PRIOR /* PgUp on arrow keypad */
CONST DIK_LEFTARROW = DIK_LEFT /* LeftArrow on arrow keypad
*/
CONST DIK_RIGHTARROW = DIK_RIGHT /* RightArrow on arrow
keypad */
CONST DIK_DOWNARROW = DIK_DOWN /* DownArrow on arrow keypad
*/
CONST DIK_PGDN = DIK_NEXT /* PgDn on arrow keypad */
/*
* Alternate names for keys originally not used on US
keyboards.
*/
CONST DIK_CIRCUMFLEX = DIK_PREVTRACK /* Japanese keyboard
*/
```


3D Programming Guide

Part

XV

A large, light gray circle is positioned to the right of the word 'Part'. Inside this circle, the letters 'XV' are written in a bold, white, sans-serif font. A horizontal gray line extends from the left side of the circle, passing behind the word 'Part'.

15 3D Programming Guide

15.1 Introduction

The DirectX 3D command set is an advanced 3D library included with the IWBASIC development environment. The 3D engine is class (OOP) based for better management of code.

To distribute programs written with the 3D command library you will need to include the dx3d9r.dll with your program. The DLL is located in the 'redist' directory of the IWBASIC installation. You will also find a copy on the CD if you purchased a CD version of IWBASIC.

Minimum requirements

- IWBASIC 2.0 or greater.
- DirectX 9.0c or greater installed.
- DirectX 9.0c capable video card.
- Windows 98, ME, 2000, XP or Vista

15.2 Classes

15.2.1 C3DCamera

Class Description:

The camera defines the view of the 3D world

Class Methods:

C3DCamera	Class destructor.
C3DCamera	Class constructor.
Create	Creates the camera.
EnableFog	Enables fog for the camera view.
Free	Frees the camera.
GetDirection	Returns the camera's direction vector.
GetLookAt	Returns the look at point.
GetPosition	Returns the camera's position.
GetUpVector	Returns the camera's up vector.
LockYAxis	Locks the Y axis of the camera.
LookAt	Orients the camera towards a vector.
Move	Moves the camera along the X/Z axis.
ObjectInView	Determines if an object is in view.
Orient	Changes the cameras up and direction vectors.
Position	Positions the camera at an absolute 3D vector.
Project	projects a 3D point in world space to screen 2D coordinates.

Rotate	Rotates a camera based on yaw, pitch and roll.
SetAspectRatio	Sets the aspect ration of the camera..
SetBackPlane	Sets the back plane of the camera..
SetFogColor	Sets the color value of fog.
SetFogRange	Sets the near and far ranges of fog.
SetFOV	Sets the cameras field of view..
SetFrontPlane	Sets the front plane of the camera..
SetMode	Sets the cameras mode, either perspective or ortho..
SetY	Sets the Y axis position of the camera..
Unproject	Transforms a 2D screen coordinate into 3D world space..

Class Member Variables:

*m_pData	void	Private data internal to the camera.
----------	------	--------------------------------------

Located in:

ebx3d.incc

Class Hierarchy[C3DCamera](#)**15.2.1.1 _C3DCamera****Syntax:**[_C3DCamera\(\)](#)**Purpose:**

The destructor frees the camera. Called when the object is deleted or goes out of scope.

Parameters:

None

Returned value(s):

None

15.2.1.2 C3DCamera**Syntax:**[C3DCamera\(\)](#)**Purpose:**

Class constructor. Called automatically by the compiler.

Parameters:

None

Returned value(s):

None

15.2.1.3 Create**Syntax:**

Create(POINTER pScreen)

Purpose:

Creates and initialized the camera object.

Parameters:

pScreen - A C3DScreen object.

Returned value(s):

None

Example:

```
C3DCamera c
c.Create(s)
c.Position(0,0,-8)
c.Orient(0,0,1,0,1,0)
c.SetBackPlane(500)
```

Additional Info:

The 3D screen must be created before using this method

15.2.1.4 EnableFog**Syntax:**

EnableFog(int toggle)

Purpose:

Enables/Disables fog for the camera.

Parameters:

toggle - TRUE to enable fog in the cameras view, FALSE to disable.

Returned value(s):

None

Example:

```
Cam1.EnableFog(TRUE)
```

Additional Info:**15.2.1.5 Free****Syntax:****Free()****Purpose:**

Frees the camera and associated memory.

Parameters:

None

Returned value(s):

None

Example:

```
cam1.Free()
```

Additional Info:

Called automatically by the destructor.

15.2.1.6 GetDirection**Syntax:****GetDirection(), VECTOR3****Purpose:**

Returns the camera's direction vector.

Parameters:

None

Returned value(s):

The direction the camera is facing in a VECTOR3 structure.

Example:

```
VECTOR3 v  
v = cam1.GetDirection()
```

Additional Info:

The returned vector is expressed in world space.

15.2.1.7 GetLookAt

Syntax:

GetLookAt(), VECTOR3

Purpose:

Returns the current point the camera is looking at.

Parameters:

None

Returned value(s):

A point in 3D space the camera is currently looking at. The point is stored in a VECTOR3 UDT.

Example:

```
look = cam1.GetLookAt()
```

15.2.1.8 GetPosition

Syntax:

GetPosition(), VECTOR3

Purpose:

Returns the current position of the camera in world space.

Parameters:

None

Returned value(s):

A VECTOR3 UDT containing the camera's positions

Example:

```
VECTOR3 v  
v = cam1.GetPosition()
```

See Also:

15.2.1.9 GetUpVector

Syntax:

GetUpVector(), VECTOR3

Purpose:

Returns the camera's up vector.

Parameters:

None

Returned value(s):

A VECTOR3 UDT containing the camera's current up vector.

Example:

```
VECTOR3 v  
v = GetUpVector();
```

15.2.1.10 LockYAxis

Syntax:

LockYAxis(int bLocked)

Purpose:

Locks the Y axis of the camera.

Parameters:

bLocked - TRUE to lock the Y axis of the camera at its current value, FALSE to allow the Y axis to change.

Returned value(s):

None

Example:

```
LockYAxis(1)
```

Additional Info:

Using a locked Y axis allows the camera to move in a first person shooter (FPS) style. The height of the camera can be set with the SetY method.

See Also:

[SetY](#)

15.2.1.11 LookAt

Syntax:

LookAt(float x, float y, float z)

Purpose:

Changes the direction vector of a camera to look at a specific point in world space.

Parameters:

x, y, z - The point to look at.

Returned value(s):

None

Application:

```
VECTOR3 v  
v = mesh1.GetPosition(TRUE)  
cam1.LookAt(v.x, v.y, v.z)
```

Additional Info:

Orientation of the camera in regards to the Y axis is unaffected by this command. If your view of the word was upside down before this command it will still be upside down looking at the specific point.

15.2.1.12 Move**Syntax:**

Move(float LeftRight, float FrontBack)

Purpose:

Moves a camera along the X/Z axis.

Parameters:

LeftRight - Distance to move camera along the X axis.

FrontBack - Distance to move camera along the Y axis.

Returned value(s):

None

Application:

```
cam1.move( -1.0 , 0)
```

Additional Info:

Positive values move the camera right and back, negative values move the camera left and front respectively. To change the Y axis use the [SetY](#) method. This method is designed to move the camera using a FPS (First Person Shooter) methodology. By locking the Y axis and using Move you can achieve the same walking results seen in popular games.

See Also:

[LockYAxis](#)

15.2.1.13 ObjectInView**Syntax:**

ObjectInView(POINTER pObject), INT

Purpose:

Tests whether an object is within the viewing frustum of a camera.

Parameters:

pObject - A C3DObject or derivatives.

Returned value(s):

TRUE if the any part of the object is within the viewing frustum of the camera. FALSE otherwise

Application:

```
if cam1.ObjectInView(mesh1)
    ShowStats()
endif
```

Additional Info:

The object must be visible for ObjectInView to return TRUE.

15.2.1.14 Orient**Syntax:**

Orient(float DirX, float DirY, float DirZ, float ux, float uy, float uz)

Purpose:

Aligns a camera so that its z-direction points along the direction vector [DirX, DirY, DirZ] and its y-direction aligns with the vector [ux, uy, uz].

Parameters:

DirX, DirY, DirZ - The new direction vector.

ux, uy, uz - The new up vector.

Returned value(s):

None

Application:

```
c.Create(s)
c.Position(0,0,-8)
c.Orient(0,0,1,0,1,0)
c.SetBackPlane(500)
```

15.2.1.15 Position**Syntax:**

Position(float x, float y, float z)

Purpose:

Positions the camera at an absolute 3D point.

Parameters:

x, y, z - The new position for the camera in world space

Returned value(s):

None

Application:

```
c.Create(s)
c.Position(0,0,-8)
c.Orient(0,0,1,0,1,0)
c.SetBackPlane(500)
```

Additional Info:

The orientation of the camera is unchanged by this command

15.2.1.16 Project**Syntax:**

Project(float x, float y, float z), VECTOR3

Purpose:

Projects a point in world space to screen coordinates based on the camera's view and projection matrices.

Parameters:

x,y,z - The point in camera world space to project to screen space.

Returned value(s):

A VECTOR3 UDT. *x* = Screen X position, *y* = Screen Y position, *z* = a Z difference between 0.0 for the front plane and 1.0 for the back plane.

Application:

```
' Use sphere's position to get associated screen coordinates
vCamProject = cam1.Project(pSpherePos.x,pSpherePos.y,pSpherePos.z)
```

See Also:

[Unproject](#)

15.2.1.17 Rotate**Syntax:**

Rotate(float yaw, float pitch, float roll)

Purpose:

Rotate the camera about all three axis allowing for 6 degrees of freedom (6DOF)

Parameters:

yaw - The angle to rotate along the Y axis.

pitch - The angle to rotate along the X axis.

roll - The angle to rotate along the Z axis.

Returned value(s):

None

Application:

```
IF KeyDown(DIK_RIGHT) then cam1.Rotate(1*.01745 * fAdjust,0,0)
```

Additional Info:

Angles are expressed in radians and are measured clockwise when looking along the rotation axis toward the origin.

15.2.1.18 SetAspectRatio

Syntax:

SetAspectRatio(float aspect)

Purpose:

Sets the aspect ratio of the camera.

Parameters:

aspect - Aspect ratio.

Returned value(s):

None

Additional Info:

The aspect ratio is defined as view space width divided by height when the camera is initially created.

15.2.1.19 SetBackPlane

Syntax:

SetBackPlane(float plane)

Purpose:

Sets the position of the back clipping plane for the camera's viewport.

Parameters:

plane - The new back clipping plane.

Returned value(s):

None

Application.

```
c.Create(s)
```

```
c.Position(0,0,-8)
c.Orient(0,0,1,0,1,0)
c.SetBackPlane(500)
```

Additional Info:

Objects positioned outside of the front and back clipping planes will not be rendered. The default back clipping plane is 100.0 when the camera is created.

15.2.1.20 SetFogColor**Syntax:**

SetFogColor(int r, int g, int b)

Purpose:

Set the color of fog, if enabled for the camera.

Parameters:

r, g, b - The new fog color. Values range from 0 to 255

Returned value(s):

None

Application:

```
cam1.SetFogColor(0,0,255)
cam1.SetFogRange(14,400)
```

Additional Info:

The default fog color is white 255, 255, 255

See Also:

[EnableFog](#), [SetFogRange](#)

15.2.1.21 SetFogRange**Syntax:**

SetFogRange(float Near, float Far)

Purpose:

Sets the near and far ranges of the fog effect.

Parameters:

Near - Depth at which fog starts to take effect.

Far - Depth at which fog effect ends.

Returned value(s):

None

Application:

```
cam1.SetFogColor(0,0,255)  
cam1.SetFogRange(14,400)
```

Additional Info:

Default range is 1.0, 1000.0. Ranges are specified as distance from the camera.

See Also:

[SetFogColor](#), [EnableFog](#)

15.2.1.22 SetFOV**Syntax:**

SetFOV(float fov)

Purpose:

Sets the field of view for a camera's viewport.

Parameters:

fov - The new field of view.

Returned value(s):

None

Application:

```
cam1.SetFOV(0.5)
```

Additional Info:

This fov value is a fraction of the viewing angle. The default value is $\pi/4$. The value must be greater than 0.

15.2.1.23 SetFrontPlane**Syntax:**

SetFrontPlane(float plane)

Purpose:

Sets the front clipping plane of the camera.

Parameters:

plane - The new front clipping plane

Returned value(s):

None

Application:

```
cam1.SetFrontPlane(5.0)
```

Additional Info:

Objects positioned outside of the front and back clipping planes will not be rendered. The default front clipping plane is 1.0 when the camera is created.

See Also:

[SetBackPlane](#)

15.2.1.24 SetMode**Syntax:**

SetMode(int cameraMode)

Purpose:

Switches between perspective and orthogonal viewing modes.

Parameters:

cameraMode - The new viewing mode.

Returned value(s):

None

Application:

```
cam1.SetMode(FALSE)
```

Additional Info:

The default viewing mode is perspective. Use FALSE for the cameraMode parameter to switch to orthogonal viewing mode.

15.2.1.25 SetY**Syntax:**

SetY(float fy)

Purpose:

Sets the Y axis position for a camera.

Parameters:

fy - The new Y axis position.

Returned value(s):

None

Application:

```
cam1.SetY(4.5)
```

Additional Info:

When using a camera with a locked Y axis the method can be used to set the axis. Used for FPS style cameras.

See Also:

[LockYAxis](#)

15.2.1.26 Unproject**Syntax:**

Unproject(float x, float y, float z), VECTOR3

Purpose:

Projects a vector from screen space into object space.

Parameters:

x, y, z - The coordinates and depth of the point to unproject.

Returned value(s):

A VECTOR3 UDT containing the point in 3D camera space corresponding to the 2D input.

Application:

```
vSelect = cam1.Unproject(s.MouseX(), s.MouseY, .5)
```

Additional Info:

On input X and Y are the screen point, Z is a value between 0.0 and 1.0 where 0.0 would correspond to the front plane and 1.0 the back plane. For example if your current backplane was set to 500.0 and frontplane of 1.0 an input Z of .5 would produce an output Z of approximately 250.0

See Also:

[Project](#)

15.2.2 C3DLandscape**Class Description:**

Class for generating multi-textured landscapes.

Class Methods:

Load	Creates a landscape from height, light and coverage files
----------------------	---

Located in:

ebx3d.incc

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

-----[C3DLandscape](#)

15.2.2.1 Load

Syntax:

Load([C3DScreen](#) pScreen, string heightMapFile, string lightMapFile, string coverageFile, string tile1File, string tile2File, string tile3File, float blockSizeX, float blockSizeZ, float maxBlockHeight, float textureScaleX, float textureScaleZ, int useGrayscaleLightMap), int

Purpose:

Creates a landscape object.

Parameters:

pScreen - Screen object.

heightMapFile - Image file containing the height map for the landscape.

lightMapFile - Image file containing the light map for the landscape.

coverageFile - Image file containing the coverage file for the landscape.

tile1File - The first blended texture.

tile2File - The second blended texture.

tile3File - The third blended texture.

blockSizeX - The size of each block of the landscape along x axis.

blockSizeZ - The size of each block of the landscape along z axis.

maxBlockHeight - The height that 255 in the height map file corresponds to.

textureScaleX - The scale of texture along the width of each block.

textureScaleZ - The scale of texture along the height of each block.

useGrayscaleLightMap - Flag that specifies whether the light map is grayscale or color.

Returned value(s):

None

Application:

```
'Setup the landscape properties
/* Size of each block of land along x and z axis */
blockSizeX = 2.0f
blockSizeZ = 2.0f
/* Tell the program what value 255 of height map color corresponds to */
maximumLandHeight = 50.0f
/* Texture scaling along x axis, fraction along the width of the texture per block of
landTextureScaleX = .2f
/* Texture scaling along z axis, fraction along the height of the texture per block of
landTextureScaleZ = .2f
```

```
/* Whether light map is gray scale or color, in this case, it's gray scale light map */
grayScaleLightMap = true
'load the landscape.
mediaPath = GetStartPath() + "media\\"
m.Load(s,mediaPath + "map.bmp",mediaPath + "lightmap.tga",mediaPath + "coverage.bmp",_
      mediaPath + "tile1.tga",mediaPath + "tile2.tga",mediaPath + "tile3.tga",_
      blockSizeX, blockSizeZ, maximumLandHeight, landTextureScaleX, landTextureScaleZ)
m.Position(-256,25,512)
m.EnableLighting(false)
m.InitCollision(false)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

-----[C3DLandscape](#)

Additional Info:

The height of each vertex in the landscape is calculated using the red component of each pixel of the height map file. `maxBlockHeight` is the height in 3D units that the value 255 corresponds to. The range of elevations will be $-\text{maxBlockHeight} < 0 < \text{maxBlockHeight}$ with 0 corresponding to a red pixel value of 127 in the height map image file.

The light map file determines the brightness of each vertex of the landscape. The light map file must have the same dimensions as the height map as each pixel in the light map determines the final brightness of the corresponding pixel in the height map. If `useGrayScaleLightMap` is TRUE then only the red component of the light map is actually used resulting in an adjusted light range of 0 for totally black to 255 for full lighting of the vertex. If `useGrayScaleLightMap` is FALSE then the color and intensity of the light map pixel is used to blend the vertex creating the effect of using a colored light source.

The coverage image file must be the same size as the height map. Each pixel value in the coverage file determines the amount of the three textures should be used to render each corner of the landscape. Red determines the blending of the tile1 texture, green determines the bending of the tile2 texture and blue determines the bending of the tile3 texture. The final pixel color before being modulated by the light map is calculated using the formula $(r / 255)*t1 + (g / 255)*t2 + (b / 255)*t3$ where r, g, b are the red, green, blue components of the coverage file pixel. t1, t2, t3 are the color of the three textures tile1, tile2, and tile3 respectively.

The dimensions of the height map file determine the number of triangles generated in the final mesh. There will be a total of $\text{width} * (\text{height}-1) * 2$ triangles generated. A height map image of 256*256 will result in a mesh of 130560 triangles for example. Internally a quadtree is used for rendering the landscape to only draw the parts of the mesh that are in the view of the camera.

15.2.3 C3DLight

Class Description:

Class used to create light objects in a scene.

Class Methods:

Create	Creates a Direct3D light source..
Disable	Disables a light in the scene.
Enable	Enable a light in a scene.
SetAmbient	Sets a light's ambient color and intensity.
SetAttenuation	Sets a light's distance attenuation.
SetDiffuse	Sets the diffuse color emitted by the light.
SetDirection	Sets the look at point of a light.
SetFalloff	Decrease in illumination between a spotlight's inner and outer cones..
SetPhi	Sets the angle, in radians, defining the outer edge of the spotlight's outer cone.
SetRange	Sets the distance beyond which the light has no effect.
SetSpecular	Sets the Specular color emitted by the light.
SetTheta	Sets the angle, in radians, of a spotlight's inner cone.

Located in:

ebx3d.incc

Class Hierarchy[C3DObject](#)-----[C3DLight](#)**15.2.3.1 Create****Syntax:****[Create](#)([POINTER](#) pScreen, int nType, int nIndex), int****Purpose:**

Creates a Direct3D light source.

Parameters:*pScreen* - A C3DScreen object.*nType* - The type of the light.*nIndex* - Zero based light index.**Returned value(s):**

TRUE if the light could be created, FALSE otherwise.

Application:

```
light.Create(s, LIGHT_POINT, 1)
light.Position(0, 20, -100)
light.SetAttenuation(0, 1/200.0, 0)
```

```
light.SetSpecular(.5, .5, .5, 1)
light.SetAmbient(.4, .4, .4, 1)

scene.CreateScene(s)
scene.AddChild(light)
scene.AddChild(m)
```

Class Hierarchy

[C3DObject](#)

-----[C3DLight](#)

Additional Info:

nType can be one of LIGHT_POINT, LIGHT_SPOT or LIGHT_DIRECTIONAL. Point lights have color and position within a scene, but no single direction. They give off light equally in all directions. Directional lights have only color and direction, not position. They emit parallel light. Spotlights have color, position, and direction in which they emit light. Light emitted from a spotlight is made up of a bright inner cone and a larger outer cone, with the light intensity diminishing between the two.

A light must be rendered with the Draw method, or drawn by its parent object, before it effects any other meshes in the scene. The order in which lighting is specified has no effect on the final lighting. A light object can be positioned, orientated, and rotated using the same methods as a C3DObject and can serve as the parent of other objects in the scene.

The nIndex value is used by Direct3D internally. Each light should have its own unique zero based index number however it is ok to have multiple lights with the same index number as long as only one of them is rendered per frame. If two lights containing the same index number are rendered then Direct3D will use the last one specified.

15.2.3.2 Disable

Syntax:

Disable()

Purpose:

Disables the light.

Parameters:

None

Returned value(s):

None

Class Hierarchy

[C3DObject](#)
-----[C3DLight](#)

See Also:

[Enable](#)

15.2.3.3 Enable

Syntax:

Enable()

Purpose:

Enables the light.

Parameters:

None

Returned value(s):

None

Class Hierarchy

[C3DObject](#)
-----[C3DLight](#)

See Also:

[Disable](#)

15.2.3.4 SetAmbient

Syntax:

SetAmbient(float r, float g, float b, float a)

Purpose:

Sets the ambient color of the light

Parameters:

r, g, b, a - Then new ambient color

Returned value(s):

None

Application:

```
light.Create(s, LIGHT_POINT, 1)
light.Position(0, 20, -100)
light.SetAttenuation(0, 1/200.0, 0)
```

```
light.SetSpecular(.5,.5,.5,1)
light.SetAmbient(.4,.4,.4,1)
```

Class Hierarchy[C3DObject](#)-----[C3DLight](#)**Additional Info:**

Ambient lighting provides constant lighting for a scene. It lights all object vertices the same because it is not dependent on any other lighting factors such as vertex normals, light direction, light position, range, or attenuation. It is the fastest type of lighting but it produces the least realistic results.

Each light object that has an ambient color adds to the total ambient lighting in the scene. Although Direct3D uses RGBA values for lights, the alpha color component is not used.

See Also:[SetDiffuse](#), [SetSpecular](#)**15.2.3.5 SetAttenuation****Syntax:****[SetAttenuation\(float att0, float att1, float att2\)](#)****Purpose:**

Specifies how the light intensity changes over distance.

Parameters:

att0 - Constant attenuation factor.

att1 - Linear attenuation factor.

att2 - Quadratic attenuation factor.

Returned value(s):

None

Application:

```
light1.SetAttenuation(0.01,0.0,0.0)
```

Class Hierarchy[C3DObject](#)-----[C3DLight](#)**Additional Info:**

The parameters can range from 0.0 to infinity but must never be negative. The attenuation of a light depends on the type of light and the distance between the light and the vertex position. To calculate attenuation, Direct3D uses the following equation:

$$\text{Atten} = 1 / (\text{att0i} + \text{att1i} * d + \text{att2i} * d^2)$$

Where *d* is the distance from vertex position to light position. If *d* is greater than the light's range, set by the `SetRange` method, Microsoft Direct3D makes no further attenuation calculations and applies no effects from the light to the vertex.

The attenuation constants act as coefficients in the formula you can produce a variety of attenuation curves by making simple adjustments to them. You can set `Attenuation1` to 1.0 to create a light that doesn't attenuate but is still limited by range, or you can experiment with different values to achieve various attenuation effects.

The attenuation at the maximum range of the light is not 0.0. To prevent lights from suddenly appearing when they are at the light range, an application can increase the light range. Or, the application can set up attenuation constants so that the attenuation factor is close to 0.0 at the light range. The attenuation value is multiplied by the red, green, and blue components of the light's color to scale the light's intensity as a factor of the distance light travels to a vertex.

See Also:

[SetRange](#)

15.2.3.6 SetDiffuse

Syntax:

SetDiffuse(float r, float g, float b, float a)

Purpose:

Sets the diffuse color of the light.

Parameters:

r, g, b, a - Then new diffuse color

Returned value(s):

None

Application:

```
light1.SetDiffuse(1.0, .5, .5)
```

Class Hierarchy

[C3DObject](#)

-----[C3DLight](#)

Additional Info:

After adjusting the light intensity for any attenuation effects, the lighting engine computes how much

of the remaining light reflects from a vertex, given the angle of the vertex normal and the direction of the incident light. The lighting engine skips to this step for directional lights because they do not attenuate over distance. The system considers two reflection types, diffuse and specular, and uses a different formula to determine how much light is reflected for each. After calculating the amounts of light reflected, Microsoft Direct3D applies these new values to the diffuse and specular reflectance properties of the current material. The resulting color values are the diffuse and specular components that the rasterizer uses to produce Gouraud shading and specular highlighting.

Although Direct3D uses RGBA values for lights, the alpha color component is not used.

The default diffuse color is 1.0, 1.0, 1.0

See Also:

[SetAmbient](#), [SetSpecular](#)

15.2.3.7 SetDirection

Syntax:

SetDirection(float x, float y, float z)

Purpose:

Changes the lights direction for spot and directional light types.

Parameters:

x, *y*, *z* - The new direction vector

Returned value(s):

None

Application:

```
light1.SetDirection(0.0, -1.0, 0.0)
```

Class Hierarchy

[C3DObject](#)

-----[C3DLight](#)

Additional Info:

Direction vectors are described as distances from a logical origin, regardless of the light's position in a scene. Therefore, a spotlight that points straight into a scene along the positive z-axis has a direction vector of <0,0,1> no matter where its position is defined to be. Similarly, you can simulate sunlight shining directly on a scene by using a directional light whose direction is <0,-1,0>. Obviously, you don't have to create lights that shine along the coordinate axes; you can mix and match values to create lights that shine at more interesting angles.

Note Although you don't need to normalize a light's direction vector, always be sure that it has

magnitude. In other words, don't use a $\langle 0,0,0 \rangle$ direction vector.

15.2.3.8 SetFalloff

Syntax:

SetFalloff(float falloff)

Purpose:

Sets the decrease in illumination between a spotlight's inner and outer cones..

Parameters:

falloff - New falloff value.

Returned value(s):

None

Application:

```
light1.SetFalloff(2.0)
```

Class Hierarchy

[C3DObject](#)

-----[C3DLight](#)

Additional Info:

Falloff values can range from +/- infinity.

The effect of various falloff values on the actual lighting is subtle, and a small performance penalty is incurred by shaping the falloff curve with falloff values other than 1.0. For these reasons, this value defaults to 1.0

15.2.3.9 SetPhi

Syntax:

SetPhi(float phi)

Purpose:

Sets the angle, in radians, defining the outer edge of a spotlight's outer cone.

Parameters:

phi - The new angle, in radians.

Returned value(s):

None

Application:

```
light1.SetPhi(3.1415 / 3.0)
```

Class Hierarchy

[C3DObject](#)

-----[C3DLight](#)

Additional Info:

A *spotlight* emits a cone of light. Only objects within the cone are illuminated. The cone produces light of two degrees of intensity, with a central brightly lit section (the *umbra*) that acts as a point source, and a surrounding dimly lit section (the *penumbra*) that merges with the surrounding deep shadow.

The phi angle is the angle of the penumbra. The allowed range of phi is [Theta, 0]. The default for theta and phi are $\pi/2$ and 0 respectively.

See Also:

[SetTheta](#)

15.2.3.10 SetRange

Syntax:

SetRange(float range)

Purpose:

Sets the range property of a spot or point light source.

Parameters:

range - The new range.

Returned value(s):

None

Application:

```
light.SetRange(500.0)
```

Class Hierarchy

[C3DObject](#)

-----[C3DLight](#)

Additional Info:

A light's range property determines the distance, in world space, at which meshes in a scene no longer receive light emitted by that object. The range parameter contains a floating-point value that represents the light's maximum range, in world space. Directional lights don't use the range

property.

See Also:

[SetAttenuation](#)

15.2.3.11 SetSpecular

Syntax:

SetSpecular(float r, float g, float b, float a)

Purpose:

Sets the specular color of the light

Parameters:

r, g, b, a - The new specular color.

Returned value(s):

None

Application:

```
light1.SetSpecular(1.0, .02, .05)
```

Class Hierarchy

[C3DObject](#)

-----[C3DLight](#)

Additional Info:

After adjusting the light intensity for any attenuation effects, the lighting engine computes how much of the remaining light reflects from a vertex, given the angle of the vertex normal and the direction of the incident light. The lighting engine skips to this step for directional lights because they do not attenuate over distance. The system considers two reflection types, diffuse and specular, and uses a different formula to determine how much light is reflected for each. After calculating the amounts of light reflected, Microsoft Direct3D applies these new values to the diffuse and specular reflectance properties of the current material. The resulting color values are the diffuse and specular components that the rasterizer uses to produce Gouraud shading and specular highlighting.

Although Direct3D uses RGBA values for lights, the alpha color component is not used.

The default specular color is 0.0, 0.0, 0.0

See Also:

[SetDiffuse](#), [SetAmbient](#)

15.2.3.12 SetTheta

Syntax:

SetTheta(float theta)

Purpose:

Sets the angle, in radians, of a spotlight's inner cone.

Parameters:

theta - The new angle, in radians.

Returned value(s):

None

Application:

```
light1.SetTheta(3.1415/4.0)
```

Class Hierarchy

[C3DObject](#)

-----[C3DLight](#)

Additional Info:

A *spotlight* emits a cone of light. Only objects within the cone are illuminated. The cone produces light of two degrees of intensity, with a central brightly lit section (the *umbra*) that acts as a point source, and a surrounding dimly lit section (the *penumbra*) that merges with the surrounding deep shadow.

The theta angle is the angle of the umbra. The allowed range of theta is [0, π]. The default for theta and phi are $\pi/2$ and 0 respectively.

See Also:

[SetPhi](#)

15.2.4 C3DMesh

Class Description:

Mesh class for loading, creating and displaying geometry in the 3D world.

Class Methods:

BeginRenderCubeTexture	Sets the render target to the texture in this mesh.
BuildOctree	Builds an octree for the mesh.
CreateBox	Creates a box mesh or cube with the dimensions specified.
CreateCubeTexture	Creates a cubic environment texture used for rendering the reflections of a scene.
CreateCylinder	Creates a cylinder mesh.
CreateMesh	Creates a custom mesh object.

CreateMeshEx	Creates a custom mesh object.
CreateRectangle	Creates a single rectangle mesh..
CreateSphere	Creates a sphere mesh..
EnableAlpha	Enables or disables alpha blending.
EnableLighting	Enables or disables lighting for this mesh.
EnableSphereMapping	Enables sphere mapping for a texture in the mesh.
EndRenderCubeTexture	Ends the process of rendering a face of a cubic environment texture..
GetID	Returns the user defined ID for this mesh.
GetIndexCount	Returns the number of indices stored in the index buffer of an object.
GetVertexCount	Returns the size of the vertex buffer used by the mesh object.
GetVertexFormat	Returns the flexible vertex format (FVF) of the mesh.
GetVertexSize	Returns the size of the vertex in bytes.
Load3DS	Loads a mesh from a 3D Studio format mesh file.
LoadMD2	Loads a mesh from a MD2 format file.
LoadSkinnedX	Loads a skinned X format mesh file..
LoadTexture	Loads and applies a texture from an image file.
LoadX	Loads an X format mesh file.
LockIndexBuffer	Locks the index buffer for direct reading/writing.
LockVertexBuffer	Locks the vertex buffer for direct reading/writing.
ReallocateMesh	Resizes the vertex and index buffers of the mesh.
RecalcBoundingBox	Recalculates the minimum bounding box of a mesh.
SetAlphaArg1	Sets the first alpha argument for the texture stage.
SetAlphaArg2	Sets the second alpha argument for the texture stage.
SetAlphaDest	Sets the destination alpha blend state..
SetAlphaOp	Sets the alpha operator for the render stage.
SetAlphaOperation	Sets the alpha operation for the texture stage.
SetAlphaSource	Sets the source alpha blend state.
SetAnimation	Sets the animation frames and times for an animated mesh.
SetAnimationMode	Sets the animation mode for an animated mesh.
SetColorArg1	Sets the first color argument for the texture stage.
SetColorArg2	Sets the second color argument for the texture stage.
SetColorOperation	Specifies the color operation used for the texture stage.
SetCulling	Sets the culling mode used when rendering the mesh.
SetFill	Sets the fill mode used when rendering the mesh.
SetID	Sets a user defined identifier.
SetMaterial	Sets the material values to use when rendering the mesh.
SetNamedAnimation	Selects a named animation set for MD2 format files.
SetShading	Sets the shading mode used when rendering the mesh.
SetVertexFormat	Specifies the vertex format (FVF) used by a custom mesh.
SetVertexSize	Specifies the vertex structure size used by a custom mesh.
SetVisible	Sets a mesh's visibility.
UnlockIndexBuffer	Unlocks the index buffer after writing.
UnlockVertexBuffer	Unlocks the vertex buffer after writing.
UpdateAllAnimations	Updates the animation frames of this mesh and all child mesh's.

UpdateAnimation	Updates the animation frames of this mesh.
UseVertexColor	Specifies whether a mesh uses vertex color instead of materials.

Located in:

ebx3d.incc

Class Hierarchy[C3DObject](#)-----[C3DMesh](#)**15.2.4.1 BeginRenderCubeTexture****Syntax:****[BeginRenderCubeTexture\(int texNum, int side, float x, float y, float z\)](#)****Purpose:**

Sets the render target of Direct3D to the cubic environment texture contained in this mesh object.

Parameters:

texNum - The texture number.

side - Which side of the texture should be used as the render target.

x, y, z - The position of the camera used to render the cube map.

Returned value(s):

None

Application:

```
'render a static cubemap
for x = 0 to 5
  sphere.BeginRenderCubeTexture(0,x,0,0,0)
  s.Clear(rgba(0,0,255,255))
  room.Draw()
  sphere.EndRenderCubeTexture()
next x
```

Class Hierarchy[C3DObject](#)-----[C3DMesh](#)**Additional Info:**

Cubic environment maps sometimes referred to as cube maps are textures that contain image data representing the scene surrounding an object, as if the object were in the center of a cube. Each face of the cubic environment map covers a 90-degree field of view in the horizontal and vertical, and there are six faces, or sides, per cube map.

After calling this command the cube map is selected as the render target and the side specified becomes the surface on which all objects are drawn. Draw your scene to each of the six sides of

the cube map in an identical manner you would if rendering to the primary surface. You should hide the object containing the cube map texture before rendering using the [SetVisible](#) method.

Rendering a cubic environment texture is time consuming since 6 separate copies of your scene must be rendered. It is more efficient to use a static cube map than a dynamic one for complex scenes.

Unlike drawing a normal scene with [C3DScreen::BeginScene](#), the [C3DScreen::Clear](#) command should be used after `BeginRenderCubeTexture`

See Also:

[EndRenderCubeTexture](#)

15.2.4.2 BuildOctree

Syntax:

BuildOctree(int maxPoly, int depth)

Purpose:

Builds an octree for the mesh.

Parameters:

maxPoly - Maximum triangle count for each node.

depth - The maximum depth of the tree.

Returned value(s):

None

Application:

```
if Mesh3.Load3DS(s, GETSTARTPATH + "park.3ds", FALSE)
    mesh3.BuildOctree(500,4)
    'initialize the collision system for this mesh
    mesh3.InitCollision(FALSE)
endif
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

An octree separates a mesh into equally sized blocks of eight. Each block will be further separated depending on the maxPoly and depth settings. Using octrees speeds up rendering of a scene by only drawing parts of the mesh that are currently in view of the camera. The original data is unchanged and multiple index buffers are used to render the correct parts of the mesh.

Octrees work best on large objects (> 1000 polys) with a depth level of 5 or less. The larger the

tree the more time it will take to determine which parts are visible. BuildOctree has no effect on landscape or skinned mesh objects.

Once an octree is built it will be used automatically when drawing the mesh. No further interaction is required.

See Also:

15.2.4.3 CreateBox

Syntax:

CreateBox([C3DScreen](#) pScreen, float width, float height, float depth, int bInside), int

Purpose:

Creates a box mesh or cube with the dimensions specified.

Parameters:

pScreen - A C3DScreen object.

width - Width of box.

height - Height of box.

depth - Depth of box.

bInside - Culling flag.

Returned value(s):

TRUE if the mesh was successfully created, FALSE otherwise.

Application:

```
'create a box to copy. Make it an 'inside' cube
temp.CreateBox(screen, width,height,depth,true)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

If *bInside* is TRUE then the box will be visible when the camera is inside of it, such as a room. You can make a box visible from both inside and out by using the [SetCulling](#) method. Note however that light reflection will be determined by the *bInside* parameter

15.2.4.4 CreateCubeTexture

Syntax:

CreateCubeTexture(int texNum, int edgeLength, string name)

Purpose:

Creates a cubic environment texture used for rendering the reflections of a scene.

Parameters:

texNum - The zero based texture number.

edgeLength - Size of the edges of all the top-level faces of the cube texture.

name - A unique name used internally by the texture manager.

Returned value(s):

None

Class Hierarchy

[C3DObject](#)

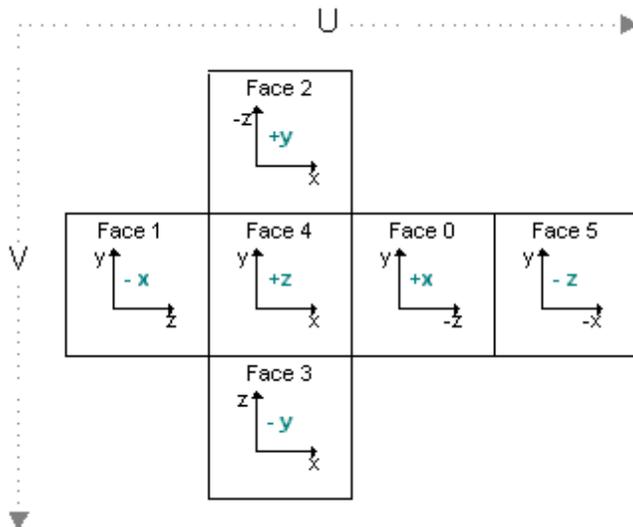
-----[C3DMesh](#)

Additional Info:

Cubic environment maps—sometimes referred to as cube maps—are textures that contain image data representing the scene surrounding an object, as if the object were in the center of a cube.

Each face of the cubic environment map covers a 90-degree field of view in the horizontal and vertical, and there are six faces per cube map.

Each face of the cube is oriented perpendicular to the x/y, y/z, or x/z plane, in world space. The following figure shows how each plane corresponds to a face.



Not all older video cards support creating cube textures. A message box will be presented to the user if their card lacks the capability. The name of the texture must be unique and is used internally by the texture manager to prevent duplicate resources from being created for the same mesh object.

Higher edge lengths result in more detailed reflections but consume more video ram. Lower edge

lengths result in blockier looking reflections but are more suitable for dynamic mapping where the reflection is recalculated every frame. If your combining textures to create semi-reflective surfaces then stick with lower edge lengths. Most video cards require edge lengths to be a power of two.

The approximate amount of video ram require for one cube map is shown below:

Edge Length	Memory
64	98K
128	393K
256	1.57MB
512	6.29MB
1024	25.2MB

Texture coordinates for the mesh containing the cube texture are calculated automatically. If creating a mesh using [CreateMesh](#) you don't need to specify an FVF constant for that texture number or include texture coordinates for the cube texture in your vertex buffer.

See Also:

[BeginRenderCubeTexture](#), [EndRenderCubeTexture](#)

15.2.4.5 CreateCylinder

Syntax:

CreateCylinder([C3DScreen](#) pScreen, float height, int segments, float rad, int bInside), int

Purpose:

Creates a cylinder mesh.

Parameters:

pScreen - A C3DScreen object.

height - The height of the cylinder.

segments - Number of polygon segments around the diameter.

rad - The radius of the cylinder.

bInside - Culling flag.

Returned value(s):

TRUE if the mesh could be created, FALSE otherwise.

Application:

```
mesh1.CreateCylinder(s, 3.0, 20, .25, FALSE)
```

Class Hierarchy

[C3DObject](#)

-----[C3D Mesh](#)

Additional Info:

Higher segment values results in a better looking cylinder but increases the poly count of the object.

If `bInside` is `TRUE` then the cylinder will be visible when the camera is inside of it. You can make a cylinder visible from both inside and out by using the [SetCulling](#) method. Note however that light reflection will be determined by the `bInside` parameter

15.2.4.6 CreateMesh

Syntax:

CreateMesh([C3D Screen](#) `pScreen`, `int numVertices`, `int numIndices`, `uint fvf`), `int`

Purpose:

Creates a custom mesh object.

Parameters:

pScreen - A `C3D Screen` object

numVertices - Number of vertices to allocate for the vertex buffer.

numIndices - Number of indices to allocate for the index buffer.

fvf - The flexible vertex format of the mesh.

Returned value(s):

`TRUE` if the mesh could be created, `FALSE` otherwise.

Application:

```
mesh1.CreateMesh(pScreen,numVertices,numIndices,D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_D:
```

Class Hierarchy

[C3D Object](#)

-----[C3D Mesh](#)

Additional Info:

The object is created as initially hidden. After setting the vertex and index buffers use [SetVisible](#) to allow the object to be rendered and [RecalcBoundingBox](#) to calculate the dimensions of the internal bounding box used for visibility determination

See Also:

[CreateMeshEx](#) , [FVF Constants](#)

15.2.4.7 CreateMeshEx

Syntax:

CreateMeshEx([C3DScreen](#) pScreen, uint primitiveType, int numVertices, int numIndices, int numPrimitives, uint fvf), int

Purpose:

Creates a custom mesh object.

Parameters:

pScreen - A C3DScreen object.

primitiveType - One of six primitive types offered by Direct3D.

numVertices - Number of vertices to allocate for the vertex buffer

numIndices - Number of Indices to allocate for the index buffer

numPrimitives - Number of primitives to render

fvf - the flexible vertex format of the mesh

Returned value(s):

TRUE if the mesh could be created, FALSE otherwise.

Application:

```
mesh1.CreateMeshEx(D3DPT_LINELIST, numVertices, numIndices, numIndices/2, D3DFVF_XYZ
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

The object is created as initially hidden. After setting the vertex and index buffers use [SetVisible](#) to allow the object to be rendered and [RecalcBoundingBox](#) to calculate the dimensions of the internal bounding box used for visibility determination

The six primitive constants allowed are:

D3DPT_POINTLIST

D3DPT_LINELIST

D3DPT_LINESTRIP

D3DPT_TRIANGLELIST

D3DPT_TRIANGLESTRIP

D3DPT_TRIANGLEFAN

See Also:

[CreateMesh](#), [FVF Constants](#)

15.2.4.8 CreateRectangle

Syntax:

CreateRectangle([C3DScreen](#) pScreen), int

Purpose:

Creates a rectangular mesh comprised of two triangles.

Parameters:

pScreen - A C3DScreen object

Returned value(s):

TRUE if the mesh could be created, FALSE otherwise.

Application:

```
Mesh2.CreateRectangle(s)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

The mesh returned has four vertices with texture coordinates of {0,0} {0,1} {1,0} {1,1} and can be used to create billboards, walls, or any other single sided object that needs a rectangular mesh. The dimensions of the rectangle are 1x1 and can be scaled to any size using the [C3DObject::Scale](#) method.

See Also:

[CreateBox](#)

15.2.4.9 CreateSphere

Syntax:

CreateSphere([C3DScreen](#) pScreen, int numBands, float rad, int bInside), int

Purpose:

Creates a sphere mesh..

Parameters:

pScreen - A C3DScreen Object.

numBands - Density of vertices around the the sphere.

rad - Radius of the sphere.

bInside - Culling flag.

Returned value(s):

TRUE if the mesh could be created, FALSE otherwise.

Application:

```
Mesh3.CreateSphere(s, 30, 0.5, FALSE)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

Higher number of bands will produce a better looking sphere at the cost of more memory.

If `bInside` is TRUE then the sphere will be visible when the camera is inside of it, such as a room. You can make a sphere visible from both inside and out by using the [SetCulling](#) command. Note however that light reflection will be determined by the `bInside` parameter.

See Also:

[CreateCylinder](#)

15.2.4.10 EnableAlpha

Syntax:

EnableAlpha(int bEnable)

Purpose:

Enables or disables alpha blending for the specified object.

Parameters:

bEnable - TRUE to enable alpha blending, FALSE to disable.

Returned value(s):

None

Application:

```
ball.EnableAlpha(TRUE)  
ball.SetAlphaOp(D3DBLENDOP_MAX)  
ball.SetAlphaSource(D3DBLEND_SRCCOLOR)  
ball.SetAlphaDest(D3DBLEND_INVSRCOLOR)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

Alpha blending is disabled by default.

See Also:

[SetAlphaOp](#), [SetAlphaSource](#), [SetAlphaDest](#)

15.2.4.11 EnableLighting**Syntax:**

EnableLighting(int bEnable)

Purpose:

Enables or disables lighting on a per mesh basis.

Parameters:

bEnable - TRUE to enable lighting for the mesh.

Returned value(s):

None

Application:

```
house.EnableLighting(TRUE)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

If lighting is disabled for an object then the color of a vertex is defined by the diffuse color, material or texture alone.

See Also:

[UseVertexColor](#)

15.2.4.12 EnableSphereMapping**Syntax:**

EnableSphereMapping(int texturelayer)

Purpose:

Enables sphere mapping for the mesh using the specified texture.

Parameters:

texturelayer - The zero based texture to use.

Returned value(s):

None.

Application:

```
' Load tiger.x mesh and enable sphere mapping on texture layer 0
tiger.LoadX(GETSTARTPATH + "tiger.x")
tiger.EnableSphereMapping(0)
tiger.Position(0,0,0)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

To turn off sphere mapping use -1 as the *texturelayer* parameter.

15.2.4.13 EndRenderCubeTexture

Syntax:**EndRenderCubeTexture()****Purpose:**

Ends the process of rendering a face of a cubic environment texture and resets the render target to the screen.

Parameters:

None.

Returned value(s):

None.

Application:

```
'render a static cubemap
for x = 0 to 5
  sphere.BeginRenderCubeTexture(0,x,0,0,0)
  s.Clear(rgba(0,0,255,255))
  room.Draw()
  sphere.EndRenderCubeTexture()
next x
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

BeginRenderCubeTexture and EndRenderCubeTexture must be used in pairs.

See Also:[BeginRenderCubeTexture](#)**15.2.4.14 GetID****Syntax:**

GetID(), int

Purpose:

Returns the user defined identifier for a mesh..

Parameters:

None.

Returned value(s):

The user defined identifier set by the SetID method.

Application:

```
type = #<C3DMesh>pMesh.GetID()
```

Class Hierarchy[C3DObject](#)-----[C3DMesh](#)**See Also:**[SetID](#)**15.2.4.15 GetIndexCount****Syntax:****[GetIndexCount\(\), int](#)****Purpose:**

Returns the number of indices stored in the index buffer of this mesh.

Parameters:

None.

Returned value(s):

The number of indices.

Application:

```
'create a box to copy. Make it an 'inside' cube
```

```
box.CreateBox(width,height,depth,true)
numVertices = box.GetVertexCount()
numIndices = box.GetIndexCount()
```

Class Hierarchy[C3DObject](#)-----[C3DMesh](#)**See Also:**[GetVertexCount](#)**15.2.4.16 GetVertexCount****Syntax:****GetVertexCount(), int****Purpose:**

Returns the size of the vertex buffer used by the mesh object.

Parameters:

None.

Returned value(s):

The number of vertices.

Application:

```
'create a box to copy. Make it an 'inside' cube
box.CreateBox(width,height,depth,true)
numVertices = box.GetVertexCount()
numIndices = box.GetIndexCount()
```

Class Hierarchy[C3DObject](#)-----[C3DMesh](#)**See Also:**[GetIndexCount](#)**15.2.4.17 GetVertexFormat****Syntax:****GetVertexFormat(), UINT****Purpose:**

Returns the flexible vertex format (FVF) of the mesh.

Parameters:

None.

Returned value(s):

The vertex format of the mesh.

Application:

```
fvf = myMesh.GetVertexFormat()  
IF fvf & D3DFVF_NORMAL = 0  
    CreateNormals(myMesh)  
ENDIF
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

See Also:

[Vertex format constants.](#)

15.2.4.18 GetVertexSize**Syntax:**

GetVertexSize(), int

Purpose:

Returns the size of the vertex in bytes.

Parameters:

None.

Returned value(s):

The size of a single vertex of the mesh.

Application:

```
sz = mymesh.GetVertexSize()
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:**See Also:**

[GetVertexCount](#)

15.2.4.19 Load3DS

Syntax:

[Load3DS](#)([C3DScreen](#) pScreen, string filename, int bSeparate), int

Purpose:

Loads a mesh from a 3D Studio format mesh file.

Parameters:

pScreen - A C3DScreen object.

filename - Filename of the 3DS file to load.

bSeparate - Whether to combine all models or not.

Returned value(s):

TRUE if the mesh could be loaded, FALSE otherwise.

Application:

```
if Mesh3.Load3DS(s, GETSTARTPATH + "park.3ds", FALSE)
    mesh3.BuildOctree(500, 4)
    'initialize the collision system for this mesh
    mesh3.InitCollision(FALSE)
endif
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

.3DS files can contain multiple model and texture references. If bSeparate is FALSE then all models in the file are combined into a single mesh while still maintaining correct orientation of the original models. If bSeparate is TRUE then the models are placed into separate objects in an internal hierarchy allowing multi texturing to work properly. Separating the models requires more memory but no extra effort to draw or position the total mesh object.

If using octrees to determine visibility then you should use bSeparate as FALSE to allow a complete octree to be constructed.

15.2.4.20 LoadMD2

Syntax:

[LoadMD2](#)([C3DScreen](#) pScreen, string filename, string texturefile), int

Purpose:

Loads a mesh from a MD2 format file.

Parameters:

pScreen - A C3DScreen object

filename - The filename of the MD2 mesh.

texture - The associated skin texture for the MD2 mesh.

Returned value(s):

TRUE if the mesh could be loaded, FALSE otherwise

Application:

```
IF (pMesh2.LoadMD2 (GETSTARTPATH + "super.md2", GETSTARTPATH + "super.bmp"))
    pMesh2.EnableLighting (false)
    pMesh2.SetNamedAnimation ("run", 2000, timeGetTime ())
    pMesh2.SetAnimationMode (ANIM_LOOP)
ENDIF
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

Frame based animation of the MD2 mesh is fully supported using the built in animation commands of the 3D library. Linear interpolation is used for in-between frame animation.

The named animations available for the [SetNamedAnimation](#) method are:

"stand", "run", "attack", "pain1", "pain2", "pain3", "jump", "death1", "death2"
 "death3", "flip", "salute", "taunt", "wave", "point", "crstand", "crwalk", "crattack"
 "crpain", "crdeath"

15.2.4.21 LoadSkinnedX**Syntax:**

LoadSkinnedX([C3DScreen](#) pScreen, string filename), int

Purpose:

Loads a skinned X format mesh file..

Parameters:

pScreen - A C3DScreen object.

filename - The skinned X file to load,

Returned value(s):

TRUE if the mesh could be loaded, FALSE otherwise

Application:

```
'load a skinned mesh
if pMesh.LoadSkinnedX(GETSTARTPATH + "tiny.x")
    pMesh.Positiont(0,0,0)
    pMesh.Orient(0,0,1,0,1,0)
    'for skinned X meshes StartFrame = animation track, EndFrame is ignored
    'if track = -1 then no animation
    pMesh.SetAnimation(0,0,800,timeGetTime())
ENDIF
```

Class Hierarchy[C3DObject](#)-----[C3DMesh](#)**15.2.4.22 LoadTexture****Syntax:****LoadTexture(int texNum, string filename, UINT clr)****Purpose:**

Loads and applies a texture from an image file.

Parameters:*texNum* - Texture number for 0 to 7.*filename*- Filename or resource name.*clr*- Transparency color key.**Returned value(s):****Application:**

```
mesh1.LoadTexture(0,GETSTARTPATH+"tile3.tga",0)
```

Class Hierarchy[C3DObject](#)-----[C3DMesh](#)**Additional Info:**

The 3D engine supports up to 8 textures per mesh. The *clr* parameter can be set to 0 to disable transparency for the texture, otherwise use the RGBA function to specify the transparency color. Alpha is significant and should usually be set to FF (255) for opaque color keys. Thus, for opaque black, the value would be equal to RGBA(0,0,0,255).

The texture engine will try an load from your programs resources first, and then try to load from a file if that fails.

The memory associated with the texture is automatically freed when the C3DMesh class is destroyed or goes out of scope.

15.2.4.23 LoadX

Syntax:

[LoadX\(C3DScreen pScreen, string filename\), int](#)

Purpose:

Loads an X format mesh file.

Parameters:

pScreen - A C3DScreen object

filename - String that specifies the filename.

Returned value(s):

Application:

```
tiger.LoadX(s, GetStartPath() + "tiger.x")
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

All the meshes in the file will be collapsed into one output mesh. If the file contains a frame hierarchy, all the transformations will be applied to the mesh.

For mesh files that do not contain effect instance information, default effect instances will be generated from the material information in the .x file.

See Also:

[LoadSkinnedX](#)

15.2.4.24 LockIndexBuffer

Syntax:

[LockIndexBuffer\(\), pointer](#)

Purpose:

Locks the index buffer for direct reading/writing.

Parameters:

None.

Returned value(s):

A pointer to the index buffer.

Application:

```
POINTER pIB
pIB = *<C3DMesh>pRet.LockIndexBuffer()
#<WORD>pIB[0] = 3
#<WORD>pIB[1] = 4
#<WORD>pIB[2] = 2
*<C3DMesh>pRet.UnlockIndexBuffer()
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

The pointer returned points to an array of WORD. Use the [GetIndexCount](#) method to get the count of words in the array. The pointer will be NULL if the mesh hasn't been allocated yet by loading from a file or creating with the [CreateMesh](#) or [CreateMeshEx](#) methods.

When working with index buffers, you are allowed to make multiple lock calls. However, you must ensure that the number of lock calls match the number of unlock calls.

Be sure the buffer is unlocked before trying to render the mesh.

See Also:

[UnlockIndexBuffer](#)

15.2.4.25 LockVertexBuffer

Syntax:

[LockVertexBuffer\(\)](#), pointer

Purpose:

Locks the vertex buffer for direct reading/writing.

Parameters:

None.

Returned value(s):

A pointer to the vertex buffer memory.

Application:

```
POINTER pVB
pVB = *<C3DMesh>pRet.LockVertexBuffer()
```

```
#<VERTEX0TEXTURE>pVB.position.x = -x
#<VERTEX0TEXTURE>pVB.position.y = -y
#<VERTEX0TEXTURE>pVB.position.z = z
#<VERTEX0TEXTURE>pVB.diffuseColor = RGBA(255,0,0,255)
#<VERTEX0TEXTURE>pVB.normal = Vec3Normalize(#<VERTEX0TEXTURE>pVB.position)
*<C3DMesh>pRet.UnlockVertexBuffer()
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

The pointer returned points to an array of UDT's matching the flexible vertex format used when the mesh was loaded or created. The pointer will be NULL if the mesh hasn't been allocated yet by loading from a file or creating with the [CreateMesh](#) or [CreateMeshEx](#) methods.

When working with vertex buffers, you are allowed to make multiple lock calls. However, you must ensure that the number of lock calls match the number of unlock calls.

Be sure the buffer is unlocked before trying to render the mesh.

See Also:

[UnlockVertexBuffer](#)

15.2.4.26 ReallocateMesh

Syntax:

ReallocateMesh(int numVertices, int numIndices, uint fvf)

Purpose:

Resizes a mesh previously created with the CreateMesh method.

Parameters:

numVertices - Number of vertices to allocate for the vertex buffer.

numIndices - Number of indices to allocate for the index buffer.

fvf - The flexible vertex format of the mesh.

Returned value(s):

None.

Application:

```
'create the mesh with a vertex format that allows position, a diffuse color and a normal
'if the mesh already exists then just reallocate the buffers
if(rest = NULL)
    mesh.CreateMesh(pScreen,numVertices,numIndices,D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_
else
    mesh.ReallocateMesh(numVertices,numIndices,D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_
endif
```

Class Hierarchy[C3DObject](#)-----[C3DMesh](#)**Additional Info:**

This method should only be used with meshes created with the [CreateMesh](#) method. All data in the vertex and index buffers is destroyed before the buffers are reallocated.

15.2.4.27 RecalcBoundingBox**Syntax:****RecalcBoundingBox()****Purpose:**

Recalculates the minimum bounding box of a mesh.

Parameters:

None.

Returned value(s):

None.

Application:

```
mesh1.RecalcBoundingBox()
```

Class Hierarchy[C3DObject](#)-----[C3DMesh](#)**Additional Info:**

The bounding box of a mesh is used for visibility determination in the current viewing frustum. The method should be called after creating or modifying the vertex buffer of a mesh.

The bounding box is automatically calculated when a mesh is loaded and transformed.

15.2.4.28 SetAlphaArg1**Syntax:****SetAlphaArg1(int texStage, UINT dwArg)****Purpose:**

Sets the first alpha argument for the specified texture stage.

Parameters:

texStage - The texture stage (0-7) that this argument applies to.

dwArg - The argument to set.

Returned value(s):

None.

Application:

```
ret.SetAlphaArg1(0, D3DTA_DIFFUSE)
ret.SetAlphaOperation(0, D3DTOP_SELECTARG1)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

Used for advanced multi texturing control. *dwArg* can be one of:

D3DTA_CURRENT - Use the result of the previous texture stage, for the 0th stage, it is the same as D3DTA_DIFFUSE

D3DTA_DIFFUSE - Use the diffuse alpha estimated from the vertices' diffuse colors in Gouraud shading process

D3DTA_SPECULAR - Use the specular alpha estimated from the vertices' diffuse colors in Gouraud shading process

D3DTA_TEXTURE - Use the texture alpha of this texture stage

The default argument is D3DTA_CURRENT. If no texture is set for this stage, the default argument is D3DTA_DIFFUSE.

See Also:

[SetAlphaArg2](#)

15.2.4.29 SetAlphaArg2

Syntax:

SetAlphaArg2(int texStage, UINT dwArg)

Purpose:

Sets the second alpha argument for the specified texture stage.

Parameters:

texStage - The texture stage (0-7) that this argument applies to.

dwArg - The argument to set.

Returned value(s):

None.

Application:

```
pass2.SetAlphaArg2(0, D3DTA_DIFFUSE)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

Used for advanced multi texturing control. dwArg can be one of:

D3DTA_CURRENT - Use the result of the previous texture stage, for the 0th stage, it is the same as D3DTA_DIFFUSE

D3DTA_DIFFUSE - Use the diffuse alpha estimated from the vertices' diffuse colors in Gouraud shading process

D3DTA_SPECULAR - Use the specular alpha estimated from the vertices' diffuse colors in Gouraud shading process

D3DTA_TEXTURE - Use the texture alpha of this texture stage

The default argument is D3DTA_TEXTURE.

See Also:

[SetAlphaArg1](#)

15.2.4.30 SetAlphaDest**Syntax:**

[SetAlphaDest\(UINT dstFactor\)](#)

Purpose:

Sets the destination alpha blend state..

Parameters:

dstFactor - The blending factor.

Returned value(s):

None.

Application:

```
pass2.EnableAlpha(TRUE)
pass2.SetAlphaOp(D3DBLENDOP_MAX)
pass2.SetAlphaSource(D3DBLEND_SRCALPHA)
```

```
pass2.SetAlphaDest(D3DBLEND_SRCALPHA)
```

Class Hierarchy[C3DObject](#)-----[C3DMesh](#)**Additional Info:**

The default is D3DBLEND_INVSRCALPHA. Source and destination blend states are used in pairs.

Altering the source and destination blend states can give the appearance of emissive objects in a foggy or dusty atmosphere. For instance, if your application models flames, force fields, plasma beams, or similarly radiant objects in a foggy environment, set the source and destination blend states to D3DBLEND_ONE.

Another application of alpha blending is to control the lighting in a 3-D scene, also called light mapping. Setting the source blend state to D3DBLEND_ZERO and the destination blend state to D3DBLEND_SRCALPHA darkens a scene according to the source alpha information. The source primitive is used as a light map that scales the contents of the frame buffer to darken it when appropriate. This produces monochrome light mapping.

You can achieve color light mapping by setting the source alpha blending state to D3DBLEND_ZERO and the destination blend state to D3DBLEND_SRCCOLOR.

See Also:[Alpha Blending Constants](#)**15.2.4.31 SetAlphaOp****Syntax:****SetAlphaOp(UINT operator)****Purpose:**

Sets the alpha operator for the render stage.

Parameters:

operator - The blending operator.

Returned value(s):

None.

Application:

```
pass2.EnableAlpha(TRUE)
pass2.SetAlphaOp(D3DBLENDOP_MAX)
pass2.SetAlphaSource(D3DBLEND_SRCALPHA)
```

```
pass2.SetAlphaDest(D3DBLEND_SRCALPHA)
```

Class Hierarchy[C3DObject](#)-----[C3DMesh](#)**Additional Info:**

The default operator is D3DBLENDOP_ADD.

See Also:[Alpha Operator Constants](#)**15.2.4.32 SetAlphaOperation****Syntax:****[SetAlphaOperation\(int texStage, UINT dwArg\)](#)****Purpose:**

Sets the alpha operation for the texture stage.

Parameters:

texStage - The texture stage (0-7) this operation applies to.

dwArg - The operation to set.

Returned value(s):

None.

Application:

```
ret.SetAlphaArg1(0, D3DTA_DIFFUSE)  
ret.SetAlphaOperation(0, D3DTOP_SELECTARG1)
```

Class Hierarchy[C3DObject](#)-----[C3DMesh](#)**Additional Info:**

The default operation for the texture is D3DTOP_MODULATE.

See Also:[Texture Blending Constants](#)**15.2.4.33 SetAlphaSource****Syntax:**

SetAlphaSource(UINT srcFactor)

Purpose:

Sets the source alpha blend state.

Parameters:

srcFactor - The blending factor.

Returned value(s):

None.

Application:

```
pass2.EnableAlpha(TRUE)
pass2.SetAlphaOp(D3DBLENDOP_MAX)
pass2.SetAlphaSource(D3DBLEND_SRCALPHA)
pass2.SetAlphaDest(D3DBLEND_SRCALPHA)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

The default is D3DBLEND_SRCALPHA. Source and destination blend states are used in pairs.

Altering the source and destination blend states can give the appearance of emissive objects in a foggy or dusty atmosphere. For instance, if your application models flames, force fields, plasma beams, or similarly radiant objects in a foggy environment, set the source and destination blend states to D3DBLEND_ONE.

Another application of alpha blending is to control the lighting in a 3-D scene, also called light mapping. Setting the source blend state to D3DBLEND_ZERO and the destination blend state to D3DBLEND_SRCALPHA darkens a scene according to the source alpha information. The source primitive is used as a light map that scales the contents of the frame buffer to darken it when appropriate. This produces monochrome light mapping.

You can achieve color light mapping by setting the source alpha blending state to D3DBLEND_ZERO and the destination blend state to D3DBLEND_SRCCOLOR.

See Also:

[Alpha Blending Constants](#)

15.2.4.34 SetAnimation

Syntax:

SetAnimation(int StartFrame, int EndFrame, float duration, float timeNow)

Purpose:

Sets the current animation sequence for MD2 and Skinned X meshes.

Parameters:

StartFrame - The first frame to be displayed

EndFrame - The last frame to be displayed

duration - How long the whole animation will last, in seconds

timeNow - The current time, in seconds

Returned value(s):

None.

Application:

```
player.SetAnimation(10,30,5.0,timeGetTime() )
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

For Skinned X meshes StartFrame = animation track, EndFrame is ignored.

See Also:

[UpdateAnimation](#), [SetNamedAnimation](#), [SetAnimationMode](#)

15.2.4.35 SetAnimationMode

Syntax:

[SetAnimationMode\(int mode\)](#)

Purpose:

Sets the animation mode for an animated mesh.

Parameters:

mode - The new mode to set.

Returned value(s):

None.

Application:

```
mesh1.SetAnimationMode(ANIM_ONCE)
```

Class Hierarchy

[C3DObject](#)

[-----C3D Mesh](#)**Additional Info:**

The default mode is ANIM_LOOP. Currently only MD2 meshes support animation modes.

See Also:

[Animation Mode Constants](#)

15.2.4.36 SetColorArg1**Syntax:**

SetColorArg1(int texStage, UINT dwArg)

Purpose:

Sets the first color argument for the texture blending stage.

Parameters:

texStage - The texture stage (0-7) that this argument applies to.

dwArg - The argument to set.

Returned value(s):

None.

Application:

```
'PASS 2
pass2.SetColorArg1(0, D3DTA_DIFFUSE)
pass2.SetColorArg2(0, D3DTA_TEXTURE)
pass2.SetColorOperation(0, D3DTOP_MODULATE)
```

Class Hierarchy

[C3DObject](#)

[-----C3D Mesh](#)

Additional Info:

Used for advanced multi texturing control. dwArg can be one of:

D3DTA_CURRENT - Use the result of the previous texture stage, for the 0th stage, it is the same as D3DTA_DIFFUSE

D3DTA_DIFFUSE - The texture argument is the diffuse color interpolated from vertex components during Gouraud shading. If the vertex does not contain a diffuse color, the default color is 0xffffffff

D3DTA_SPECULAR - The texture argument is the specular color interpolated from vertex components during Gouraud shading. If the vertex does not contain a specular color, the default color is 0xffffffff

D3DTA_TEXTURE - The texture argument is the texture color for this texture stage.

The default argument is `D3DTA_CURRENT`. If no texture is set for this stage, the default argument is `D3DTA_DIFFUSE`.

See Also:

`SetColorArg2`

15.2.4.37 `SetColorArg2`

Syntax:

`SetColorArg2(int texStage, UINT dwArg)`

Purpose:

Sets the second color argument for the texture blending stage.

Parameters:

texStage - The texture stage (0-7) that this argument applies to.

dwArg - The argument to set.

Returned value(s):

None.

Application:

```
'PASS 2
pass2.SetColorArg1(0, D3DTA_DIFFUSE)
pass2.SetColorArg2(0, D3DTA_TEXTURE)
pass2.SetColorOperation(0, D3DTOP_MODULATE)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

Used for advanced multi texturing control. *dwArg* can be one of:

`D3DTA_CURRENT` - Use the result of the previous texture stage, for the 0th stage, it is the same as `D3DTA_DIFFUSE`

`D3DTA_DIFFUSE` - The texture argument is the diffuse color interpolated from vertex components during Gouraud shading. If the vertex does not contain a diffuse color, the default color is `0xffffffff`.

`D3DTA_SPECULAR` - The texture argument is the specular color interpolated from vertex components during Gouraud shading. If the vertex does not contain a specular color, the default color is `0xffffffff`.

`D3DTA_TEXTURE` - The texture argument is the texture color for this texture stage.

The default argument is D3DTA_TEXTURE.

See Also:

[SetColorArg1](#)

15.2.4.38 SetColorOperation

Syntax:

SetColorOperation(int texStage, UINT dwArg)

Purpose:

Sets the color operation for the texture stage.

Parameters:

texStage - The texture stage (0-7) this operation applies to.

dwArg - The operation to set.

Returned value(s):

None.

Application:

```
'PASS 2
pass2.SetColorArg1(0, D3DTA_DIFFUSE)
pass2.SetColorArg2(0, D3DTA_TEXTURE)
pass2.SetColorOperation(0, D3DTOP_MODULATE)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

The default operation for the texture is D3DTOP_MODULATE.

See Also:

[Texture Blending Constants](#)

15.2.4.39 SetCulling

Syntax:

SetCulling(int mode)

Purpose:

Sets the culling mode for the mesh.

Parameters:

mode - The new culling mode.

Returned value(s):

None.

Application:

```
cube.SetCulling(D3DCULL_NONE)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

The default culling mode is D3DCULL_CCW. The culling modes define how back faces are culled when rendering a mesh.

See Also:

[Culling Flags](#)

15.2.4.40 SetFill**Syntax:**

SetFill(int mode)

Purpose:

Sets the fill mode for the mesh.

Parameters:

mode - The new fill mode.

Returned value(s):

None.

Application:

```
statue.SetFill(FILL_WIREFRAME)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

The default fill mode is FILL_SOLID.

See Also:

[Mesh Fill Styles](#)**15.2.4.41 SetID****Syntax:****[SetID\(int id\)](#)****Purpose:**

Sets a user defined identifier for the mesh.

Parameters:

id - A 32 bit value.

Returned value(s):

None.

Application:

```
bumper2.SetID(99)
```

Class Hierarchy[C3DObject](#)

-----[C3DMesh](#)

See Also:[GetID](#)**15.2.4.42 SetMaterial****Syntax:****[SetMaterial\(D3DMATERIAL material\)](#)****Purpose:**

Sets the material values to use when rendering the mesh.

Parameters:

material - A UDT of type D3DMATERIAL

Returned value(s):

None.

Application:

```
D3DMATERIAL mat
mat.Diffuse = RGBAtoD3DC( RGBA(255,0,0,255) )
mat.Specular = RGBAtoD3DC( RGBA(255,255,0,255) )
```

```
mat.Ambient = RGBAtoD3DC( RGBA(0,0,0,0) )
mat.Emissive = RGBAtoD3DC( RGBA(0,0,0,0) )
mat.Power = 0.0
moon.SetMaterial(mat)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

Materials describe how polygons reflect light or appear to emit light in a 3-D scene. Material properties detail a material's diffuse reflection, ambient reflection, light emission, and specular highlight characteristics.

The Diffuse and Ambient members of the D3DMATERIAL UDT describe how a material reflects the ambient and diffuse light in a scene. Because most scenes contain much more diffuse light than ambient light, diffuse reflection plays the largest part in determining color. Additionally, because diffuse light is directional, the angle of incidence for diffuse light affects the overall intensity of the reflection. Diffuse reflection is greatest when the light strikes a vertex parallel to the vertex normal. As the angle increases, the effect of diffuse reflection diminishes.

The Emissive member of the D3DMATERIAL UDT is used to describe the color and transparency of the emitted light. Emission affects an object's color and can, for example, make a dark material brighter and take on part of the emitted color.

Specular reflection creates highlights on objects, making them appear shiny. The D3DMATERIAL UDT contains two members that describe the specular highlight color as well as the material's overall shininess. You establish the color of the specular highlights by setting the Specular member to the desired RGBA color the most common colors are white or light gray. The values you set in the Power member control how sharp the specular effects are.

15.2.4.43 SetNamedAnimation

Syntax:

SetNamedAnimation(string name, float duration, float timeNow)

Purpose:

Sets a named animation for MD2 format meshes.

Parameters:

name - The name of the animation track.

duration - How long the animation should take from the first frame to the last.

timeNow - The current time, in seconds.

Returned value(s):

None.

Application:

```
IF (pMesh2.LoadMD2 (GETSTARTPATH + "super.md2", GETSTARTPATH + "super.bmp"))
    pMesh2.EnableLighting (false)
    pMesh2.SetNamedAnimation ("run", 2000, timeGetTime ())
    pMesh2.SetAnimationMode (ANIM_LOOP)
ENDIF
```

Class Hierarchy[C3DObject](#)-----[C3DMesh](#)**Additional Info:**

The named animations available are as follows:

"stand", "run", "attack", "pain1", "pain2", "pain3", "jump", "death1", "death2"
"death3", "flip", "salute", "taunt", "wave", "point", "crstand", "crwalk", "crattack"
"crpain", "crdeath"

Not all MD2 meshes support all animation names. There may be additional names available provided by the author of the mesh. Setting the named animation automatically loads the start and end frames for that animation track.

MD2 animations use linear interpolation between frames to smooth animation sequences.

See Also:[UpdateAnimation](#)**15.2.4.44 SetShading****Syntax:****SetShading(int mode)****Purpose:**

Sets the shading mode for the mesh.

Parameters:

mode - The new shading mode.

Returned value(s):

None.

Application:

```
sphere.SetShading (SHADE_GOURAUD)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

The default shading mode is SHADE_FLAT.

The shading mode used to render a polygon has a profound effect on its appearance. Shading modes determine the intensity of color and lighting at any point on a polygon face.

In the flat shading mode, the Direct3D rendering pipeline renders a polygon, using the color of the polygon material at its first vertex as the color for the entire polygon. 3-D objects that are rendered with flat shading have visibly sharp edges between polygons if they are not coplanar.

When Direct3D renders a polygon using Gouraud shading, it computes a color for each vertex by using the vertex normal and lighting parameters. Then, it interpolates the color across the face of the polygons. The interpolation is done linearly.

See Also:

[Mesh Shading Styles](#)

15.2.4.45 SetVertexFormat

Syntax:

SetVertexFormat(uint fvf)

Purpose:

Specifies the vertex format (FVF) used by a custom mesh.

Parameters:

fvf - The flexible vertex format flags.

Returned value(s):

None.

Application:

```
mesh1.SetVertexFormat(D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_DIFFUSE)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

Microsoft Direct3D applications can define model vertices in several different ways. Support for flexible vertex definitions, also known as flexible vertex formats or flexible vertex format codes,

makes it possible for your application to use only the vertex components it needs, eliminating those components that aren't used. By using only the needed vertex components, your application can conserve memory and minimize the processing bandwidth required to render models.

This method is used internally by CreateMesh and CreateMeshEx. There is little reason to call this method directly.

See Also:

[FVF Constants](#)

15.2.4.46 SetVertexSize

Syntax:

SetVertexSize(int nSize)

Purpose:

Specifies the vertex structure size used by a custom mesh.

Parameters:

nSize - The size of a single vertex, in bytes.

Returned value(s):

None.

Application:

```
Mesh1.SetVertexSize(len(VERTEX1TEXTURE))
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

This method is used internally by CreateMesh and CreateMeshEx. There is little reason to call this method directly.

See Also:

[SetVertexFormat](#)

15.2.4.47 SetVisible

Syntax:

SetVisible(int bVisible)

Purpose:

Sets the visibility of the mesh.

Parameters:

bVisible - TRUE to render the mesh, FALSE otherwise.

Returned value(s):

None.

Application:

```
mymesh.SetVisible(TRUE)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

Custom meshes created with CreateMesh and CreateMeshEx are initially invisible. Use this method to allow the mesh to draw.

15.2.4.48 UnlockIndexBuffer**Syntax:****UnlockIndexBuffer()****Purpose:**

Unlocks an index buffer.

Parameters:

None.

Returned value(s):

None.

Application:

```
POINTER pIB
pIB = *<C3DMesh>pRet.LockIndexBuffer()
#<WORD>pIB[0] = 3
#<WORD>pIB[1] = 4
#<WORD>pIB[2] = 2
*<C3DMesh>pRet.UnlockIndexBuffer()
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

When working with index buffers, you are allowed to make multiple lock calls. However, you must

ensure that the number of lock calls match the number of unlock calls.

Be sure the buffer is unlocked before trying to render the mesh.

See Also:

[LockIndexBuffer](#)

15.2.4.49 UnlockVertexBuffer

Syntax:

UnlockVertexBuffer()

Purpose:

Unlocks the vertex buffer.

Parameters:

None.

Returned value(s):

None.

Application:

```
*<C3DMesh>pRet.UnlockVertexBuffer()
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

When working with vertex buffers, you are allowed to make multiple lock calls. However, you must ensure that the number of lock calls match the number of unlock calls.

Be sure the buffer is unlocked before trying to render the mesh.

See Also:

[LockVertexBuffer](#)

15.2.4.50 UpdateAllAnimations

Syntax:

UpdateAllAnimations(float timeNow)

Purpose:

Updates the animations of this mesh and all child meshes.

Parameters:

timeNow - The current time, in seconds.

Returned value(s):

None.

Application:

```
scene.UpdateAllAnimations(timeGetTime() )
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

Animations are interpolated between the start time and the current time.

See Also:

[UpdateAnimation](#)

15.2.4.51 UpdateAnimation

Syntax:

UpdateAnimation(float timeNow)

Purpose:

Updates the animation for this mesh.

Parameters:

timeNow - Current time, in seconds.

Returned value(s):

None.

Application:

```
player.UpdateAnimation(timeGetTime() )
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

Additional Info:

Animations are interpolated between the start time and the current time. This method requires that an animation track has been specified with either the [SetAnimation](#) or [SetNamedAnimation](#)

methods.

See Also:

[UpdateAllAnimations](#)

15.2.4.52 UseVertexColor

Syntax:

UseVertexColor(int bValue)

Purpose:

Specify whether a mesh uses vertex coloring or material coloring..

Parameters:

bValue - True to use the color specified in the vertex, FALSE to use the material set for the mesh.

Returned value(s):

None.

Application:

```
sphere.UseVertexColor(TRUE)
```

Class Hierarchy

[C3DObject](#)

-----[C3DMesh](#)

See Also:

[SetMaterial](#)

15.2.5 C3DObject

Class Description:

Base 3D object class. Used for creating scenes, transforms and controlling the position of derived objects.

Class Methods:

C3DObject	Class destructor.
AddChild	Adds a child object.
C3DObject	Class constructor.
CreateScene	Creates a scene object.
CreateTransform	Creates a transformation object.
Draw	Draws this object and all attached child objects.
Free	Frees the object.

GetCollisionPoint	Returns the point, in 3D space, of two collided objects.
GetDirection	Retrieves the direction vector of the object.
GetMatrix	Retrieves the transformation matrix used by the object.
GetPosition	Retrieves the position of the object.
InitCollision	Initializes collision testing for this object.
LookAt	Orients the object towards a point in 3D space.
ObjectCollided	Collision testing of the object.
Orient	Sets the up and direction vector of the object.
Position	Positions the object.
RayCollided	Collision testing of the object.
Rotate	Sets the object rotation angle.
Scale	Sets the scale factor for the object.
SetMatrix	Sets the transformation matrix used by this object.
SphereCollided	Collision testing of the object.

Class Member Variables:

*m_pData	void	Private data internal to the object.
*m_pScreen	C3DScreen	The parent screen of this object.

Located in:

ebx3d.incc

Class Hierarchy[C3DObject](#)**15.2.5.1 _C3DObject****Syntax:**[_C3DObject\(\)](#)**Purpose:**

The default destructor does nothing. Call the Free method to destroy this object and all attached children.

Parameters:

None

Returned value(s):

None.

Class Hierarchy[C3DObject](#)

15.2.5.2 AddChild

Syntax:

AddChild([C3DObject](#) pChild)

Purpose:

Adds a child object to the list of objects to be drawn and transformed.

Parameters:

pChild - A C3DObject or derived class to add.

Returned value(s):

None.

Application:

```
scene.CreateScene (scrn1)
scene.AddChild (light)
scene.AddChild (m)
```

Class Hierarchy

[C3DObject](#)

Additional Info:

A scene consists of objects connected in a hierarchy. When a child is added to a parent object the child will be positioned relative to its parent allowing complex scenes to be built. Once an object is added to a parent only the parent object needs to be freed with the Free method. Rendering the parent object with Draw will also draw all of its children.

The object must have been created with [CreateScene](#), [CreateTransform](#), or be a derived class such as C3DMesh that has validly loaded geometry.

15.2.5.3 C3DObject

Syntax:

C3DObject()

Purpose:

Class constructor. Called automatically by the compiler.

Parameters:

None.

Returned value(s):

None.

Class Hierarchy[C3DObject](#)**15.2.5.4 CreateScene****Syntax:****[CreateScene\(C3DScreen pScreen\)](#)****Purpose:**

Creates a scene object. The scene object can have an unlimited number of children.

Parameters:

pScreen - The parent C3DScreen.

Returned value(s):**Application:**

```
scene.CreateScene (scrn1)
scene.AddChild (light)
scene.AddChild (m)
```

Class Hierarchy[C3DObject](#)**Additional Info:**

Objects are maintained in a hierarchy. The scene object serves as the master parent for all objects. You can have many scene objects in your program as needed.

See Also:[AddChild](#)**15.2.5.5 CreateTransform****Syntax:****[CreateTransform\(C3DScreen pScreen\)](#)****Purpose:**

Creates a transformation object.

Parameters:

pScreen - The parent C3DScreen object.

Returned value(s):

None.

Application:

```
scene.CreateScene (scrn1)
planet.CreateSphere (scrn1, 25, 1, false)
moon.createSphere (scrn1, 15, .3, false)
trans.CreateTransform (scrn1)
scene.AddChild (planet)
scene.AddChild (light)
planet.AddChild (trans)
trans.AddChild (moon)
moon.Position (3, 0, 0)
```

Class Hierarchy

[C3DObject](#)

Additional Info:

A transformation object is a non drawn entity that allows creating pivot points in your scene. By adding child objects to the transform you can position, orient, and rotate the children as one unit.

See Also:

[AddChild](#)

15.2.5.6 Draw

Syntax:

Draw()

Purpose:

Renders this object and all attached child objects. Must be used after a call to [C3DScreen::BeginScene](#).

Parameters:

None.

Returned value(s):

None.

Application:

```
scrn1.Clear (RGBA (0, 0, 0, 255))
scrn1.BeginScene (cam1)
scene.Draw ()
scrn1.RenderScene ()
```

Class Hierarchy

[C3DObject](#)

15.2.5.7 Free

Syntax:

Free()

Purpose:

Frees the internal memory used by this object and all attached child objects.

Parameters:

None.

Returned value(s):

None.

Application:

```
scene.Free()
```

Class Hierarchy

[C3DObject](#)

Additional Info:

Free does not destroy the class object, just the internal Direct3D objects attached to the class. Free is called by the destructor of C3DObject when it is destroyed or goes out of scope.

15.2.5.8 GetCollisionPoint

Syntax:

GetCollisionPoint(), VECTOR3

Purpose:

Returns the point of collision in world space.

Parameters:

None.

Returned value(s):

A VECTOR3 UDT containing the collision point

Application:

```
VECTOR3 v  
v = bumper.GetCollisionPoint()
```

Class Hierarchy

[C3DObject](#)

Additional Info:

[RayCollided](#) will set an exact collision point. [ObjectCollided](#) sets the average collision point of the two colliding triangles. [SphereCollided](#) sets the average of the three vertices of the collided triangle.

This method only has meaning in derived mesh classes where `InitCollision` has been called.

See Also:

[InitCollision](#)

15.2.5.9 GetDirection

Syntax:

[GetDirection\(\)](#), VECTOR3

Purpose:

Retrieves the direction vector of the object.

Parameters:

None.

Returned value(s):

A VECTOR3 UDT containing the direction vector.

Application:

```
v = player.GetDirection()
```

Class Hierarchy

[C3DObject](#)

Additional Info:

All objects have position, direction, and orientation of Y axis (up vector). These three vectors form the basis of rotation and position in the 3D world. The directional vector is returned as a unit vector meaning it has the length of 1. When an object is first created it is assumed to have a directional vector of 0,0,1 or facing away from the viewer. As you rotate your object the direction vector changes to represent to correct orientation of the object.

15.2.5.10 GetMatrix

Syntax:

[GetMatrix\(int bWorld\)](#), MATRIX4

Purpose:

Returns an objects transformation matrix.

Parameters:

bWorld - TRUE for world transformation matrix, FALSE for local.

Returned value(s):

The objects matrix stored in a MATRIX4 UDT.

Application:

```
MATRIX4 m  
m = mesh.GetMatrix(TRUE)
```

Class Hierarchy

[C3DObject](#)

See Also:

[SetMatrix](#)

15.2.5.11 GetPosition

Syntax:

GetPosition(int bWorld), VECTOR3

Purpose:

Retrieves the position of the object in world or local space.

Parameters:

bWorld - World space flag.

Returned value(s):

A VECTOR3 UDT containing the x,y and z position of the object.

Application:

See the example program 'world_local.iwb'

Class Hierarchy

[C3DObject](#)

Additional Info:

If *bWorld* is TRUE then the position returned is expressed in world space coordinates. If *bWorld* is FALSE then the position returned is expressed relative to the parent object.

15.2.5.12 InitCollision

Syntax:

InitCollision(int bChildren)

Purpose:

Initializes collision detection for a mesh.

Parameters:

bChildren - TRUE to also initialize collision detection for all attached children.

Returned value(s):

None.

Application:

```
sphere.InitCollision(FALSE)
```

Class Hierarchy

[C3DObject](#)

Additional Info:

Collisions between objects are performed using search trees that must be initialized before any collisions can be tested. Initializing collisions on large meshes can take time and should be done outside of the main rendering loop. Scale is ignored when generating the collision trees.

The built in collision commands will not work with skinned X format meshes as the location and orientation of the internal mesh frames are constantly changing. You can use a sphere to represent the skinned X mesh and use [SphereCollided](#) for basic collision testing as an alternative.

If *bChildren* is TRUE then all child objects attached to this mesh will be prepared for collision testing as well. This is needed for 3DS files that are separated when loaded.

See Also:

[RayCollided](#), [ObjectCollided](#)

15.2.5.13 LookAt**Syntax:**

LookAt(float x, float y, float z)

Purpose:

Orients the object towards a point in 3D space.

Parameters:

x,y,z - The point in world space to look at.

Returned value(s):

None.

Application:

```
VECTOR3 v
```

```
v = mesh1.GetPosition(TRUE)
mesh2.LookAt(v.x,v.y,v.z)
```

Class Hierarchy

[C3DObject](#)

Additional Info:

This method reorients the object so the directional vector points at the look at point, performing the necessary rotations. When an object is first created it is assumed to have a directional vector of 0,0,1 or facing away from the viewer.

See Also:

[Orient](#)

15.2.5.14 ObjectCollided

Syntax:

ObjectCollided([C3DObject](#) pTarget, int bChildren), int

Purpose:

Checks for collisions between meshes.

Parameters:

pTarget - A C3DObject object or derivatives.

bChildren - TRUE to test for collisions between any attached child objects and the target.

Returned value(s):

TRUE if this object has collided with the target object, FALSE otherwise.

Application:

```
mesh1.InitCollision()
mesh2.InitCollision()
...
if mesh1.ObjectCollided(mesh2, FALSE)
    HandleCollisions(mesh1, mesh2)
endif
```

Class Hierarchy

[C3DObject](#)

Additional Info:

For collision testing to work the InitCollision method must have been called on all objects involved in the test. This method is only valid for objects that contain validly loaded or created geometry, such as C3DObject.

The base class method does nothing, unless bChildren is specified in which case all child objects that contain a mesh will be collision tested against the target object.

See Also:

[InitCollision](#), [RayCollided](#), [SphereCollided](#), [GetCollisionPoint](#)

15.2.5.15 Orient**Syntax:**

Orient(float dx, float dy, float dz, float ux, float uy, float uz)

Purpose:

Aligns an object so that its z-direction points along the direction vector [dx, dy, dz] and its y-direction aligns with the vector [ux, uy, uz].

Parameters:

dx, dy, dz - New directional vector.

ux, uy, uz - New Up vector.

Returned value(s):

None.

Application:

```
mesh1.Orient(0,0,1, 0,1,0)
```

Class Hierarchy

[C3DObject](#)

Additional Info:

The default orientation of an object is d[0,0,1], u[0,1,0]

See Also:

[LookAt](#)

15.2.5.16 Position**Syntax:**

Position(float x, float y, float z)

Purpose:

Positions an object.

Parameters:

x, y, z - The new position for the object.

Returned value(s):

None.

Application:

```
Mesh1.Position(10,10,10)
```

Class Hierarchy

[C3DObject](#)

Additional Info:

An objects position is relative to its parent, if any.

See Also:

[GetPosition](#)

15.2.5.17 RayCollided**Syntax:**

RayCollided([VECTOR3](#) origin, [VECTOR3](#) direction, int bChildren), int

Purpose:

Collision tests an object against a ray.

Parameters:

origin - The origin of the ray.

direction - The direction of the ray.

Returned value(s):

TRUE if any part of this object, or its children, intersect the ray.

Application:

```
if mesh1.RayCollided(origin, dir, TRUE) then HandleCollisions()
```

Class Hierarchy

[C3DObject](#)

Additional Info:

The rays position and direction are specified in world space.

For collision testing to work the InitCollision method must have been called on all objects involved in the test. This method is only valid for objects that contain validly loaded or created geometry, such as C3DMesh.

The base class method does nothing, unless bChildren is specified in which case all child objects that contain a mesh will be collision tested against the ray.

See Also:

[SphereCollided](#), [GetCollisionPoint](#), [InitCollision](#), [ObjectCollided](#)

15.2.5.18 Rotate

Syntax:

Rotate(float x, float y, float z)

Purpose:

Rotates an object about all three axes.

Parameters:

x - Rotation, in radians, for the x axis.

y - Rotation, in radians, for the y axis.

z - Rotation, in radians, for the z axis.

Returned value(s):

None.

Application:

```
mesh1.Rotate(1*.01745 * fAdjust,0,0)
```

Class Hierarchy

[C3DObject](#)

Additional Info:

Rotations are performed relative to the parent object, if any. Quaternions are used internally to eliminate gimbal lock.

See Also:

[Orient](#)

15.2.5.19 Scale

Syntax:

Scale(float x, float y, float z)

Purpose:

Scales an object.

Parameters:

x - The new x scaling.

y - The new y scaling.

z - The new z scaling.

Returned value(s):

None.

Application:

```
'make our mesh twice the size  
mesh.Scale(2.0, 2.0, 2.0)
```

Class Hierarchy

[C3DObject](#)

Additional Info:

Scaling an object is done by the rendering engine. The original vertices of the mesh remain unchanged.

15.2.5.20 SetMatrix

Syntax:

[SetMatrix](#)([MATRIX4](#) mat)

Purpose:

Sets an objects transformation matrix.

Parameters:

mat - A MATRIX4 UDT.

Returned value(s):

None.

Application:

```
mesh1.SetMatrix(mat)
```

Class Hierarchy

[C3DObject](#)

Additional Info:

For advanced use.

See Also:

[GetMatrix](#)

15.2.5.21 SphereCollided

Syntax:

[SphereCollided](#)([VECTOR3](#) position, float radius, int bChildren), int

Purpose:

Collision tests an object against a sphere.

Parameters:

position - The spheres position, in world space.

radius - The spheres radius.

Returned value(s):

TRUE if any part of the object, or its children, intersect the sphere.

Application:

```
if mesh1.SphereCollided(pos, 10.0, TRUE) then HandleCollisions()
```

Class Hierarchy

[C3DObject](#)

Additional Info:

For collision testing to work the InitCollision method must have been called on all objects involved in the test. This method is only valid for objects that contain validly loaded or created geometry, such as C3DMesh.

The base class method does nothing, unless bChildren is specified in which case all child objects that contain a mesh will be collision tested against the target sphere.

See Also:

[InitCollision](#), [RayCollided](#), [ObjectCollided](#), [GetCollisionPoint](#),

15.2.6 C3DScreen

Class Description:

3D screen class.

Class Methods:

C3DScreen	Class destructor.
Begin2D	Begin rendering 2D elements.
BeginScene	Begins rendering a scene with the specified camera object.
C3DScreen	Class constructor.
Clear	Clears the 3D rendering target.
CloseScreen	Closes the 3D screen.
CreateFullScreen	Creates a Direct3D screen.
CreateWindowed	Creates a Direct3D window..
End2D	Ends rendering of 2D elements.
MouseX	Returns the current mouse X coordinate when using full screen.
MouseY	Returns the current mouse Y coordinate when using full screen.
RenderScene	Renders all drawn objects to the screen.
RenderText	Draws 2D text on the 3D screen.
Reset	Resets the size and depth of the backbuffer.
SetFont	Sets the font used for rendering 2D text.
SetRestoreCallback	Specifies the callback function used when full screen is restored.

Class Member Variables:

m_frameStartTime	int	Internal use, for calculating FPS.
m_frameCount	int	Internal use, for calculating FPS.
m_fps	int	Internal use, for calculating FPS.
m_accumulatedFrameTime	int	Internal use, for calculating FPS.
*m_pData	void	Private internal data for the screen object.
m_bFullScreen	int	TRUE if created in full screen mode.

Located in:

ebx3d.incc

Class Hierarchy

CWindow

-----[C3DScreen](#)**15.2.6.1 _C3DScreen****Syntax:**[_C3DScreen\(\)](#)**Purpose:**

Class destructor.

Parameters:

None.

Returned value(s):

None.

Class Hierarchy[C3DScreen](#)**15.2.6.2 Begin2D****Syntax:**[Begin2D\(\)](#)**Purpose:**

Begin rendering 2D elements.

Parameters:

None.

Returned value(s):

None.

Application:

```
s.Clear(rgba(0,0,255,255))
s.BeginScene(c)
sphere.Draw()
'tell DirectX we are going to draw 2D elements
s.Begin2D()
sprite1.Draw()
s.End2D()
'End of 2D stuff
s.RenderText(0,10,"FPS:"+STR$(fps),rgba(255,255,0,255))
s.RenderText(0,30,"Use arrow keys to move around",rgba(255,255,0,255))
fps = s.RenderScene()
```

Class Hierarchy

[C3DScreen](#)

Additional Info:

Call this method before drawing 3D sprites. This method must be called after BeginScene.

See Also:

[BeginScene](#).

15.2.6.3 BeginScene**Syntax:**

[BeginScene\(C3DCamera pCamera\)](#)

Purpose:

Prepares Direct3D to render objects using the specified camera.

Parameters:

pCamera - The camera to render the scene with.

Returned value(s):

None.

Application:

```
s.Clear(rgba(0,0,0,255))
s.BeginScene(c)
scene.Draw()
s.RenderText(0,10,"FPS:"+STR$(fps),rgba(255,255,0,255))
s.RenderText(0,30,"Use arrow keys to move around",rgba(255,255,0,255))
fps = s.RenderScene()
```

Class Hierarchy

[C3DScreen](#)

Additional Info:

You can have as many cameras in your program as needed. Each BeginScene must be paired with a RenderScene. Drawing of objects, text, lights, etc must be done between BeginScene and RenderScene.

See Also:

[RenderScene](#)

15.2.6.4 C3DScreen

Syntax:

C3DScreen()

Purpose:

Class Constructor.

Parameters:

None.

Returned value(s):

None.

Class Hierarchy

[C3DScreen](#)

15.2.6.5 Clear

Syntax:

Clear(UINT clr)

Purpose:

Clears the back buffer of the 3D surface to the specified color.

Parameters:

clr - The color in RGBA format.

Returned value(s):

None.

Application:

```
s.Clear( RGBA(0,0,255,255) )
s.BeginScene(c)
```

Class Hierarchy[C3DScreen](#)**Additional Info:**

This method should be called before BeginScene.

See Also:[BeginScene](#), [RGBA](#)**15.2.6.6 CloseScreen****Syntax:****CloseScreen()****Purpose:**

Closes the Direct3D screen.

Parameters:

None.

Returned value(s):

None.

Application:

```
Scene.Free()  
screen.CloseScreen()
```

Class Hierarchy[C3DScreen](#)**Additional Info:**

All objects and cameras must be freed before using this command. An object that is a child of another does not need to be freed separately.

This method is called automatically when the screen object is destroyed or goes out of scope.

15.2.6.7 CreateFullScreen**Syntax:****CreateFullScreen(int width, int height, int bPP, int vSync, opt handler=0 as UINT), int****Purpose:**

Creates a full screen exclusive Direct3D screen.

Parameters:

width - width of the screen

height - height of the screen

bpp - Bits per pixel. Valid values are 8, 16, and 32.

vsync - Vertical sync flag.

handler - Optional message handler.

Returned value(s):

0 if screen was successfully created. $\neq 0$ on failure.

Application:

```
C3DScreen s  
s.CreateFullScreen(screen_width,screen_height,32,true)
```

Class Hierarchy

[C3DScreen](#)

Additional Info:

The vsync flag indicates whether Direct3D should synchronize presenting of the rendered surface with the vertical refresh of the monitor. Set to TRUE to wait for vsync or FALSE to immediately show changes. Waiting for vsync will throttle your maximum FPS to the refresh rate of the monitor.

The optional handler allows using a standard windows handler with your full screen program to process messages sent to the internal window. This allows seamless integration standard event driven code with linear 3D code.

If screen creation fails you can determine the reason for the failure by building a debug executable and running your program under a debug session. The reason for failure will be printed in the Debug View of the IDE.

A windowed or full screen Direct3D screen must be created before any objects, cameras or meshes.

See Also:

[CreateWindowed](#)

15.2.6.8 CreateWindowed**Syntax:**

```
CreateWindowed(int l, int t, int width, int height, int style, string title, pointer  
parent, int vSync, opt handler=0 as UINT), int
```

Purpose:

Creates a windowed Direct3D screen.

Parameters:

l, t, width, height - Position and dimensions of new window.

style - Style flags of the window.

title - Caption text for window.

parent - Windows parent or NULL.

vsync - Vertical sync flag.

handler - Optional message handler.

Returned value(s):

0 if screen was successfully created. <> 0 on failure.

Application:

```
C3DScreen s
s.CreateWindowed(0,0,640,480,@CAPTION|@SIZE,"3D Test - ESC exits",NULL,false)
```

Class Hierarchy

[C3DScreen](#)

Additional Info:

The vsync flag indicates whether Direct3D should synchronize presenting of the rendered surface with the vertical refresh of the monitor. Set to TRUE to wait for vsync or FALSE to immediately show changes. Waiting for vsync will throttle your maximum FPS to the refresh rate of the monitor.

The optional handler allows using a standard windows handler with your program to process messages sent to the internal window. This allows seamless integration standard event driven code with linear 3D code.

If screen creation fails you can determine the reason for the failure by building a debug executable and running your program under a debug session. The reason for failure will be printed in the Debug View of the IDE.

A windowed or full screen Direct3D screen must be created before any objects, cameras or meshes.

See Also:

[CreateFullScreen](#)

15.2.6.9 End2D

Syntax:

End2D()

Purpose:

Ends the drawing of 3D sprites to the surface.

Parameters:

None.

Returned value(s):

None.

Application:

```
s.Clear(rgba(0,0,255,255))
s.BeginScene(c)
sphere.Draw()
'tell DirectX we are going to draw 2D elements
s.Begin2D()
sprite1.Draw()
s.End2D()
'End of 2D stuff
s.RenderText(0,10,"FPS:"+STR$(fps),rgba(255,255,0,255))
s.RenderText(0,30,"Use arrow keys to move around",rgba(255,255,0,255))
fps = s.RenderScene()
```

Class Hierarchy

[C3DScreen](#)

Additional Info:

Begin2D and End2D must be used in pairs after BeginScene has been called.

See Also:

[Begin2D](#)

15.2.6.10 MouseX

Syntax:

MouseX(), int

Purpose:

Returns the current mouse X coordinate when using full screen.

Parameters:

None

Returned value(s):

The current mouse X position.

Application:

```
x = screen.MouseX()
```

Class Hierarchy[C3DScreen](#)**Additional Info:**

Convenience method for retrieving the mouse position.

15.2.6.11 MouseY**Syntax:****MouseY(), int****Purpose:**

Returns the current mouse Y coordinate when using full screen.

Parameters:

None.

Returned value(s):

The current mouse Y position.

Application:

```
x = screen.MouseY()
```

Class Hierarchy[C3DScreen](#)**Additional Info:**

Convenience method for retrieving the mouse position.

15.2.6.12 RenderScene**Syntax:****RenderScene(opt POINTER destrect = NULL), int****Purpose:**

Renders the scene to the Direct3D surface.

Parameters:

destrect - Optional clipping rectangle.

Returned value(s):

The current frames per second.

Application:

```
s.Clear( RGBA(0,0,0,255) )
```

```
s.BeginScene(c)
scene.Draw()
s.RenderText(0,10,"FPS:"+STR$(fps),RGBA(255,255,0,255))
s.RenderText(0,30,"Use arrow keys to move around",RGBA(255,255,0,255))
fps = s.RenderScene()
```

Class Hierarchy

[C3DScreen](#)

Additional Info:

destrect is a WINRECT UDT that can clip the output of RenderScene to a specific area of the screen.

See Also:

[BeginScene](#)

15.2.6.13 RenderText

Syntax:

RenderText(int x, int y, string text, UINT col)

Purpose:

Draws text on the 3D surface.

Parameters:

x, *y* - 2D coordinate to position text.

text - The text to draw.

col - The color of the text in RGBA format.

Returned value(s):

None.

Application:

```
s.Clear(RGBA(0,0,0,255))
s.BeginScene(c)
scene.Draw()
s.RenderText(0,10,"FPS:"+STR$(fps),RGBA(255,255,0,255))
s.RenderText(0,30,"Use arrow keys to move around",RGBA(255,255,0,255))
fps = s.RenderScene()
```

Class Hierarchy

[C3DScreen](#)

Additional Info:

Text is drawn with the font specified with the [SetFont](#) method. Alpha blending is enabled for text and can be controlled using the RGBA function.

15.2.6.14 Reset

Syntax:

Reset(int width, int height, int bPP, int bFullScreen), int

Purpose:

Resets the dimensions of a Direct3D screen.

Parameters:

width - The new screen width.

height - The new screen height.

bPP - Not currently used.

bFullScreen - Not currently used. Must be FALSE.

Returned value(s):

If the screen was successfully resized the return value is TRUE, FALSE on failure.

Class Hierarchy

[C3DScreen](#)

Additional Info:

This method works with windowed Direct3D screens only. Its purpose is to resize the Direct3D surface to match the dimensions of the client area of a window.

15.2.6.15 SetFont

Syntax:

SetFont(string typeface, int height, int weight, opt int flags = 0)

Purpose:

Sets the font used for the RenderText method.

Parameters:

typeface - Name of new font.

height - Size of font in points.

weight - Weight of font.

flags - Style flags for font and character set.

Returned value(s):

None.

Application:

```
s.SetFont("Arial", 12, 400)
```

Class Hierarchy

[C3DScreen](#)

Additional Info:

This method is similar to the [SetFont](#) command used for windows.

See Also:

[RenderText](#)

15.2.6.16 SetRestoreCallback

Syntax:

SetRestoreCallback(UINT lpfn, pointer pParam)

Purpose:

Specifies the callback function used when full screen mode is restored.

Parameters:

lpfn - Address of the callback function.

pParam - User defined parameter passed to the callback.

Returned value(s):

None.

Application:

```
m = CreatePyramid(s,2)
'set our callback function and a parameter that will be passed
'when the device has been lost, such as an alt-tab switch
s.SetRestoreCallback(&restore_screen,m)
...
SUB restore_screen(pointer pParam)
    CreatePyramid(null,2,pParam)
RETURN
ENDSUB
```

Class Hierarchy

[C3DScreen](#)

Additional Info:

The callback function is called by the 3D engine when the screen buffers have been lost, usually due to an alt-tab switch to the desktop and back to your full screen program. At this time custom meshes and environment mapped textures will need to be recreated.

The format of the callback function is a subroutine with a single pointer parameter.

15.2.7 C3DSprite

Class Description:

A class for drawing 2D images on the 3D screen.

Class Methods:

_C3DSprite	Class destructor.
C3DSprite	Class constructor.
Draw	Renders the sprite.
Free	Free the sprite.
GetAngle	Returns the current rotation angle.
GetFrame	Returns the current animation frame.
GetModulateColor	Returns the modulation color and alpha value.
GetPosition	Returns the current position.
GetRotationCenter	Returns the rotation center point.
GetScaleFactor	Returns the scaling factor.
Load	Loads a sprite from an image file.
SetAngle	Sets the current rotation angle.
SetFrame	Sets the current animation frame of the sprite.
SetModulateColor	Sets the modulation color and alpha value.
SetPosition	Moves the sprite to a new position.
SetRotationCenter	Sets the rotation center point.
SetScaleFactor	Sets the scaling factor..

Class Member Variables:

*m_pData	void	Private internal data for the sprite
*m_pScreen	C3DScreen	A pointer to the 3D screen the sprite belongs to.

Located in:

ebx3d.incc

Class Hierarchy

[C3DSprite](#)

15.2.7.1 [_C3DSprite](#)

Syntax:

[_C3DSprite\(\)](#)

Purpose:

Class destructor. Called when the object is deleted or goes out of scope.

Parameters:

None.

Returned value(s):

None.

Class Hierarchy

[C3DSprite](#)

Additional Info:

The destructor will free the sprite if it hasn't been done manually.

15.2.7.2 C3DSprite

Syntax:

C3DSprite()

Purpose:

Class constructor.

Parameters:

None.

Returned value(s):

None.

Class Hierarchy

[C3DSprite](#)

15.2.7.3 Draw

Syntax:

Draw()

Purpose:

Renders the sprite.

Parameters:

None.

Returned value(s):

None.

Application:

```
s.Clear( RGBA(0,0,255,255) )
s.BeginScene(c)
sphere.Draw()
'tell DirectX we are going to draw 2D elements
s.Begin2D()
sprite1.Draw()
s.End2D()
```

```
'End of 2D stuff
s.RenderText(0,10,"FPS:"+STR$(fps),RGBA(255,255,0,255))
s.RenderText(0,30,"Use arrow keys to move around",RGBA(255,255,0,255))
fps = s.RenderScene()
```

Class Hierarchy[C3DSprite](#)**Additional Info:**

Sprites must be drawn between calls to [C3DScreen::Begin2D](#) and [C3DScreen::End2D](#)

15.2.7.4 Free**Syntax:****Free()****Purpose:**

Frees the internal memory and texture used by the sprite.

Parameters:

None.

Returned value(s):

None.

Class Hierarchy[C3DSprite](#)**Additional Info:**

The destructor of the C3DSprite class calls this method when the class is destroyed or goes out of scope. You can call the Free method to clear the sprite for reuse with another sprite image.

15.2.7.5 GetAngle**Syntax:****GetAngle(), float****Purpose:**

Returns the rotation angle of the sprite.

Parameters:

None.

Returned value(s):

The rotation angle of the sprite, in radians.

Application:

```
angle = sprite.GetAngle()
```

Class Hierarchy

[C3DSprite](#)

15.2.7.6 GetFrame**Syntax:**

GetFrame(), int

Purpose:

Returns the current animation frame of the sprite.

Parameters:

None.

Returned value(s):

The current animation frame.

Application:

```
sprite.SetFrame(sprite.GetFrame() + 1)
```

Class Hierarchy

[C3DSprite](#)

15.2.7.7 GetModulateColor**Syntax:**

GetModulateColor(), UINT

Purpose:

Returns the current modulation color.

Parameters:

None.

Returned value(s):

The modulation color in RGBA format.

Class Hierarchy

[C3DSprite](#)

Additional Info:

The texture used to draw a sprite can be tinted and alpha drawn by using a modulation color. The

default modulation color is 0xFFFFFFFF which corresponds to RGBA(255,255,255,255) which maintains the original source color and alpha values.

See Also:

[Set ModulateColor](#)

15.2.7.8 GetPosition

Syntax:

GetPosition(), VECTOR2

Purpose:

Returns the current position of the sprite.

Parameters:

None.

Returned value(s):

The sprites current position stored in a VECTOR2 UDT. Coordinates are in screen space.

Application:

```
VECTOR2 v
v = sprite.GetPosition()
sprite.SetPosition(v.x + 1, v.y -2)
```

Class Hierarchy

[C3DSprite](#)

See Also:

[SetPosition](#)

15.2.7.9 GetRotationCenter

Syntax:

GetRotationCenter(), VECTOR2

Purpose:

Returns the current center of rotation of the sprite.

Parameters:

None.

Returned value(s):

The rotation center in a VECTOR2 UDT.

Application:

```
VECTOR2 v
v = sprite.GetRotationCenter()
```

Class Hierarchy[C3DSprite](#)**Additional Info:**

The rotation center is relative the to the sprites upper left corner. The default rotation center is 0, 0

See Also:[SetRotationCenter](#).**15.2.7.10 GetScaleFactor****Syntax:****[GetScaleFactor\(\)](#), [VECTOR2](#)****Purpose:**

Returns the current scaling of the sprite.

Parameters:

None.

Returned value(s):

The sprites scale stored in a VECTOR2 UDT.

Class Hierarchy[C3DSprite](#)**Additional Info:**

The default scaling factors are 1.0, 1.0

See Also:[SetScaleFactor](#)**15.2.7.11 Load****Syntax:****[Load\(C3DScreen pScreen, string filename, opt int width = 0 , opt int height = 0 , opt int frames = 0 , opt UINT colorKey = 0xFFFFFFFF \)](#), int****Purpose:**

Loads a sprite from an image file.

Parameters:

pScreen - A C3DScreen object.

filename - Fully qualified pathname to the texture used by the sprite.

width - The width of one frame of animation contained in the texture.

height - The height of one frame of animation contained in the texture.

frames - The number of animation frames contained in the texture.

colorKey - A transparency color key for the sprite. To use the alpha channel of the texture leave this parameter at its default.

Returned value(s):

1 on success, -1 on failure.

Application:

```
'the file contains two sprites with 8 frames of animation each.  
'each frame is 64x64 pixels  
cavemen.Load(s,GetStartPath + "media\\cavemen.dds",64,64,16)
```

Class Hierarchy

[C3DSprite](#)

Additional Info:

If the entire texture is to be used for the sprite then width, height and frames can be set to 0.

15.2.7.12 SetAngle

Syntax:

SetAngle(float angle), float

Purpose:

Sets the rotation angle of the sprite.

Parameters:

angle - The rotation angle, in radians.

Returned value(s):

None.

Application:

```
cavemen.SetAngle(startTime/1000.0f)
```

Class Hierarchy

[C3DSprite](#)

Additional Info:

Sprite is rotated about the xy point set by [SetRotationCenter](#).

15.2.7.13 setFrame

Syntax:

SetFrame(int frame), int

Purpose:

Sets the animation frame of the sprite.

Parameters:

frame - The zero based animation frame.

Returned value(s):

None.

Application:

```
cavemen.SetFrame(cavemen.GetFrame() + 1)
```

Class Hierarchy

[C3DSprite](#)

See Also:

[GetFrame](#).

15.2.7.14 SetModulateColor

Syntax:

SetModulateColor(UINT clr), UINT

Purpose:

Sets the modulation color of the sprite.

Parameters:

clr - The new modulation color.

Returned value(s):

None.

Application:

```
cavemen.SetModulateColor(rgba(255,255,255,128))
```

Class Hierarchy

[C3DSprite](#)

Additional Info:

The texture used to draw a sprite can be tinted and alpha drawn by using a modulation color. The default modulation color is 0xFFFFFFFF which corresponds to rgba(255,255,255,255) which

maintains the original source color and alpha values.

See Also:

[GetModulationColor](#)

15.2.7.15 SetPosition

Syntax:

SetPosition(float x, float y)

Purpose:

Sets the position of the sprite.

Parameters:

x, y - The new position of the sprite, in screen space.

Returned value(s):

None.

Application:

```
VECTOR2 v
v = sprite.GetPosition()
sprite.SetPosition(v.x + 1, v.y -2)
```

Class Hierarchy

[C3DSprite](#)

See Also:

[GetPosition](#)

15.2.7.16 SetRotationCenter

Syntax:

SetRotationCenter(float x, float y)

Purpose:

Sets the rotation center of the sprite.

Parameters:

x, y - The new center of rotation.

Returned value(s):

None.

Application:

```
spr2.SetRotationCenter(10,10)
```

Class Hierarchy[C3DSprite](#)**Additional Info:**

The rotation center is relative the to the sprites upper left corner. The default rotation center is 0, 0

See Also:[GetRotationCenter](#).**15.2.7.17 SetScaleFactor****Syntax:****SetScaleFactor(float x, float y)****Purpose:**

Sets the scaling factor of the sprite.

Parameters:

x, y - The new scaling.

Returned value(s):

None.

Application:

```
'Make sprite twice the size
sprite.SetScaleFactor(2.0,2.0)
```

Class Hierarchy[C3DSprite](#)**Additional Info:**

The default scaling factors are 1.0, 1.0

See Also:[GetScaleFactor](#)**15.3 Structures****15.3.1 D3DCOLORVALUE****Declaration:**

```
TYPE D3DCOLORVALUE
```

```
float r
```

```
float g
```

```
float b
```

```
float a
ENDTYPE
```

15.3.2 D3DMATERIAL

```
Declaration:
TYPE D3DMATERIAL
  D3DCOLORVALUE Diffuse
  D3DCOLORVALUE Ambient
  D3DCOLORVALUE Specular
  D3DCOLORVALUE Emissive
  float Power
ENDTYPE
```

15.3.3 MATRIX4

```
Declaration:
TYPE MATRIX4
  float m[4,4]
ENDTYPE
```

15.3.4 VECTOR2

```
Declaration:
TYPE VECTOR2
  float x
  float y
ENDTYPE
```

15.3.5 VECTOR3

```
Declaration:
TYPE VECTOR3
  float x
  float y
  float z
ENDTYPE
```

15.3.6 VECTOR4

```
Declaration:
TYPE VECTOR4
  float x
  float y
  float z
  float w
ENDTYPE
```

15.3.7 VERTEX0TEXTURE

Declaration:

```
TYPE VERTEX0TEXTURE
  VECTOR3 position
  VECTOR3 normal
  UINT diffuseColor
ENDTYPE
```

Purpose:

Describes a vector with a position, normal and diffuse color. Used for direct reading/writing vertex buffers.

15.3.8 VERTEX1TEXTURE

Declaration:

```
TYPE VERTEX1TEXTURE
  VECTOR3 position
  VECTOR3 normal
  UINT diffuseColor
  VECTOR2 texCoords[1]
ENDTYPE
```

Purpose:

Describes a vector with a position, normal, diffuse color and 1 set of texture coordinates. Used for direct reading/writing vertex buffers.

15.3.9 VERTEX2TEXTURE

Declaration:

```
TYPE VERTEX2TEXTURE
  VECTOR3 position
  VECTOR3 normal
  UINT diffuseColor
  VECTOR2 texCoords[2]
ENDTYPE
```

Purpose:

Describes a vector with a position, normal, diffuse color and 2 sets of texture coordinates. Used for direct reading/writing vertex buffers.

15.3.10 VERTEX3TEXTURE

Declaration:

```
TYPE VERTEX3TEXTURE
```

```
VECTOR3 position
VECTOR3 normal
UINT diffuseColor
VECTOR2 texCoords[3]
ENDTYPE
```

Purpose:

Describes a vector with a position, normal, diffuse color and 3 sets of texture coordinates. Used for direct reading/writing vertex buffers.

15.3.11 VERTEX4TEXTURE**Declaration:**

```
TYPE VERTEX1TEXTURE
  VECTOR3 position
  VECTOR3 normal
  UINT diffuseColor
  VECTOR2 texCoords[4]
ENDTYPE
```

Purpose:

Describes a vector with a position, normal, diffuse color and 4 sets of texture coordinates. Used for direct reading/writing vertex buffers.

15.3.12 VERTEX5TEXTURE**Declaration:**

```
TYPE VERTEX1TEXTURE
  VECTOR3 position
  VECTOR3 normal
  UINT diffuseColor
  VECTOR2 texCoords[5]
ENDTYPE
```

Purpose:

Describes a vector with a position, normal, diffuse color and 5 sets of texture coordinates. Used for direct reading/writing vertex buffers.

15.3.13 VERTEX6TEXTURE**Declaration:**

```
TYPE VERTEX1TEXTURE
  VECTOR3 position
  VECTOR3 normal
  UINT diffuseColor
  VECTOR2 texCoords[6]
```

ENDTYPE

Purpose:

Describes a vector with a position, normal, diffuse color and 6 sets of texture coordinates. Used for direct reading/writing vertex buffers.

15.3.14 VERTEX7TEXTURE

Declaration:

```
TYPE VERTEX1TEXTURE
  VECTOR3 position
  VECTOR3 normal
  UINT diffuseColor
  VECTOR2 texCoords[7]
ENDTYPE
```

Purpose:

Describes a vector with a position, normal, diffuse color and 7 sets of texture coordinates. Used for direct reading/writing vertex buffers.

15.3.15 VERTEX8TEXTURE

Declaration:

```
TYPE VERTEX1TEXTURE
  VECTOR3 position
  VECTOR3 normal
  UINT diffuseColor
  VECTOR2 texCoords[8]
ENDTYPE
```

Purpose:

Describes a vector with a position, normal, diffuse color and 8 sets of texture coordinates. Used for direct reading/writing vertex buffers.

15.4 Functions

15.4.1 RGBA

Syntax:

Return = RGBA(int r, int g, int b, int a)

Purpose:

Creates a 32 bit color value used by Direct3D

Parameters:

r, *g*, *b*, *a* - The red, green, blue and alpha components.

Returned value:

The 32 bit color value

Application:

```
s.Clear( RGBA(255, 0, 255, 255) )
```

Additional Info:

For most Direct3D applications the alpha component is significant where 0 is fully transparent and 255 is fully opaque.

15.4.2 Vec3Add

Syntax:

Vec3Add(VECTOR3 A, VECTOR3 B), VECTOR3

Purpose:

Adds two 3D vectors.

Parameters:

A - The first 3D vector.

B - The second 3D vector.

Returned value:

The sum of the two 3D vectors stored in a VECTOR3 UDT.

Application:

```
v3 = Vec3Add(v1, v2)
```

15.4.3 Vec3Cross

Syntax:

Vec3Cross(VECTOR3 A, VECTOR3 B), VECTOR3

Purpose:

Determines the cross-product of two 3D vectors.

Parameters:

A - The first 3D vector.

B - The second 3D vector.

Returned value:

The cross product of the two 3D vectors stored in a VECTOR3 UDT.

Application:

```
v3 = Vec3Cross(v1, v2)
```

15.4.4 Vec3Dot

Syntax:

Vec3Dot([VECTOR3 A](#), [VECTOR3 B](#)), [VECTOR3](#)

Purpose:

Determines the dot-product of two 3D vectors.

Parameters:

A - The first 3D vector.

B - The second 3D vector.

Returned value:

The dot product of the two 3D vectors stored in a VECTOR3 UDT.

Application:

```
v3 = Vec3Dot(v1, v2)
```

15.4.5 Vec3Length

Syntax:

Vec3Length([VECTOR3 v](#)), [FLOAT](#)

Purpose:

Returns the length of a 3D vector.

Parameters:

v - vector to determine length.

Returned value:

The vector's length.

Application:

```
float length  
length = Vec3Length(v)
```

15.4.6 Vec3Lerp

Syntax:

Vec3Lerp([VECTOR3](#) v1, [VECTOR3](#) v2, [FLOAT](#) s), [VECTOR3](#)

Purpose:

Performs a linear interpolation between two 3D vectors.

Parameters:

v1 - First vector

v2 - Second vector

s - Parameter that linearly interpolates between the vectors.

Returned value:

The new vector in a VECTOR3 UDT

Application:

```
'take 10 secs to move the object between the start and end points.  
newpos = Vec3Lerp(vstart, vend, (timeGetTime() - StartTime) / 10.0)  
mesh1.SetPosition(newpos)
```

Additional Info:

This function performs the linear interpolation based on the following formula: $\text{Result} = V1 + s(V2 - V1)$.

15.4.7 Vec3Normalize

Syntax:

Vec3Normalize([VECTOR3](#) v), [VECTOR3](#)

Purpose:

Returns the normalized version of a 3D vector.

Parameters:

v - The vector to normalize.

Returned value:

The normalized vector in a VECTOR3 UDT.

Application:

```
v1 = Vec3Normalize(v1)
```

Additional Info:

Normalizing a vector returns the vector with unit length, keeping the same direction.

15.4.8 Vec3Sub

Syntax:

Vec3Sub(VECTOR3 A, VECTOR3 B), VECTOR3

Purpose:

Subtracts two 3D vectors.

Parameters:

A - The first 3D vector.

B - The second 3D vector.

Returned value:

The difference of the two 3D vectors stored in a VECTOR3 UDT.

Application:

```
v3 = Vec3Sub(v1, v2)
```

15.4.9 Vec4Add

Syntax:

Vec4Add(VECTOR4 A, VECTOR4 B), VECTOR4

Purpose:

Adds two 4D vectors.

Parameters:

A - The first 4D vector.

B - The second 4D vector.

Returned value:

The sum of the two 4D vectors stored in a VECTOR4 UDT.

Application:

```
v3 = Vec4Add(v1, v2)
```

15.4.10 Vec4Cross

Syntax:

Vec4Cross(VECTOR4 U, VECTOR4 V, VECTOR4 W), VECTOR4

Purpose:

Determines the cross-product in four dimensions.

Parameters:

U - The first 4D vector.

V - The second 4D vector.

W - The third 4D vector

Returned value:

The cross product of the three 4D vectors stored in a VECTOR4 UDT.

Application:

```
v4 = Vec4Cross(v1, v2, v3)
```

15.4.11 Vec4Dot

Syntax:

Vec4Dot([VECTOR4 A](#), [VECTOR4 B](#)), [VECTOR4](#)

Purpose:

Determines the dot-product of two 4D vectors.

Parameters:

A - The first 4D vector.

B - The second 4D vector.

Returned value:

The dot product of the two 4D vectors stored in a VECTOR4 UDT.

Application:

```
v3 = Vec4Dot(v1, v2)
```

15.4.12 Vec4Length

Syntax:

Vec4Length([VECTOR4 v](#)), [FLOAT](#)

Purpose:

Returns the length of a 4D vector.

Parameters:

v - vector to determine length.

Returned value:

The vector's length.

Application:

```
float length  
length = Vec4Length(v)
```

15.4.13 Vec4Lerp

Syntax:

Vec4Lerp(VECTOR4 v1, VECTOR4 v2, FLOAT s), VECTOR4

Purpose:

Performs a linear interpolation between two 4D vectors.

Parameters:

v1 - First vector

v2 - Second vector

s - Parameter that linearly interpolates between the vectors.

Returned value:

The new vector in a VECTOR4 UDT

Application:

```
'take 10 secs to move the object between the start and end points.  
newpos = Vec4Lerp(vstart, vend, (timeGetTime() - StartTime) / 10.0)
```

Additional Info:

This function performs the linear interpolation based on the following formula: $\text{Result} = V1 + s(V2 - V1)$.

15.4.14 Vec4Normalize

Syntax:

Vec4Normalize(VECTOR4 v), VECTOR4

Purpose:

Returns the normalized version of a 4D vector.

Parameters:

v - The vector to normalize.

Returned value:

The normalized vector in a VECTOR4 UDT.

Application:

```
v1 = Vec4Normalize(v1)
```

Additional Info:

Normalizing a vector returns the vector with unit length, keeping the same direction.

15.4.15 Vec4Sub

Syntax:

Vec4Sub([VECTOR4 A](#), [VECTOR4 B](#)), [VECTOR4](#)

Purpose:

Subtracts two 4D vectors.

Parameters:

A - The first 4D vector.

B - The second 4D vector.

Returned value:

The difference of the two 4D vectors stored in a VECTOR4 UDT.

Application:

```
v3 = Vec4Sub(v1, v2)
```

15.4.16 MatrixIdentity

Syntax:

MatrixIdentity([MATRIX4 mat](#))

Purpose:

Creates an identity matrix.

Parameters:

mat - The matrix to set.

Returned value:

None.

Application:

```
MATRIX4 mat  
MatrixIdentity(mat)
```

Additional Info:

The identity matrix is a matrix in which all coefficients are 0 except the [1,1][2,2][3,3][4,4] coefficients, which are set to 1. The identity matrix is special in that when it is applied to vertices, they are unchanged. The identity matrix is used as the starting point for matrices that will modify vertex values to create rotations, translations, and any other transformations that can be represented by a 4 × 4 matrix.

15.4.17 MatrixTranslation

Syntax:

MatrixTranslation(MATRIX4 mat, FLOAT x, FLOAT y, FLOAT z)

Purpose:

Applies the specified offsets to a matrix.

Parameters:

mat - Matrix to apply the translation to.

x, y, z - Coordinate offsets.

Returned value:

None.

Application:

```
MATRIX4 mat
MatrixIdentity(mat)
MatrixTranslation(mat, 25.0, 10.0, -2)
```

15.4.18 MatrixRotation

Syntax:

MatrixRotation(MATRIX4 mat, FLOAT x, FLOAT y, FLOAT z)

Purpose:

Applies the specified rotations to the matrix.

Parameters:

mat - Matrix to apply rotations to.

x, y, z - The rotation angles, in radians.

Returned value:

None.

Application:

```
MATRIX4 mat
MatrixIdentity(mat)
MatrixTranslation(mat, 25.0, 10.0, -2)
MatrixRotation(mat, 90 * 0.01745, 0, 0)
```

15.4.19 MatrixMultiply

Syntax:

MatrixMultiply(MATRIX4 out, MATRIX4 m1, MATRIX4 m2)

Purpose:

Determines the product of two matrices.

Parameters:

out - Matrix to copy the result to.

m1 - First matrix

m2 - Second matrix

Returned value:

None.

Application:

```
MATRIX4 matpos, matrot, matresult
MatrixIdentity(matpos)
MatrixIdentity(matrot)
MatrixTranslation(matpos, 25.0, 10.0, -2)
MatrixRotation(matrot, 90 * 0.01745, 0, 0)
MatrixMultiply(matresult, matpos, matrot)
```

Additional Info:

MatrixMultiply is used to combine the transformations of separate matrices into a resultant matrix.

15.5 Global_Constants

15.5.1 Alpha Blending Constants

Constants used by [C3DMesh::SetAlphaSource](#) and [C3DMesh::SetAlphaDest](#)

Constant	Description
D3DBLEND_BLENDFACTOR	Constant color blending factor used by the frame-buffer blender.
D3DBLEND_DESTALPHA	Blend factor is (Ad, Ad, Ad, Ad).
D3DBLEND_DESTCOLOR	Blend factor is (Rd, Gd, Bd, Ad).
D3DBLEND_INVBLENDFACTOR	Inverted constant color-blending factor used by the frame-buffer blender.
D3DBLEND_INVDESTALPHA	Blend factor is (1 - Ad, 1 - Ad, 1 - Ad, 1 - Ad).
D3DBLEND_INVDESTCOLOR	Blend factor is (1 - Rd, 1 - Gd, 1 - Bd, 1 - Ad).
D3DBLEND_INVSRCALPHA	Blend factor is (1 - As, 1 - As, 1 - As, 1 - As).
D3DBLEND_INVSRCOLOR	Blend factor is (1 - Rs, 1 - Gs, 1 - Bs, 1 - As).
D3DBLEND_ONE	Blend factor is (1, 1, 1, 1).
D3DBLEND_SRCALPHA	Blend factor is (As, As, As, As).
D3DBLEND_SRCALPHASAT	Blend factor is (f, f, f, 1); f = min(As, 1 - Ad).
D3DBLEND_SRCCOLOR	Blend factor is (Rs, Gs, Bs, As).
D3DBLEND_ZERO	Blend factor is (0, 0, 0, 0).

15.5.2 Alpha Operator Constants

Constants used by [C3DMesh::SetAlphaOp](#)

Constant	Description
D3DBLENDOP_ADD	The result is the destination added to the source.
D3DBLENDOP_MAX	The result is the maximum of the source and destination.
D3DBLENDOP_MIN	The result is the minimum of the source and destination.
D3DBLENDOP_REVSUBTRACT	The result is the source subtracted from the destination.
D3DBLENDOP_SUBTRACT	The result is the source subtracted from the destination.

15.5.3 Animation Modes

Constants used with [C3DMesh::SetAnimationMode](#)

Constant	Description
ANIM_LOOP	Continually run the animation.
ANIM_ONCE	Run the animation once.
ANIM_PINGPONG	Run the animation from start to end and back to start.
ANIM_STOP	Stop all animations.

15.5.4 Culling Flags

Constants used with the [C3DMesh::SetCulling](#) method

Constant	Description
CULL_CCW	Cull back faces with counterclockwise vertices.
CULL_CW	Cull back faces with clockwise vertices.
CULL_NONE	Do not cull back faces.

15.5.5 Flexible Vertex Format Constants

Constants used to describe mesh vertex formats.

Constant	Description
D3DFVF_DIFFUSE	Vertex format includes a diffuse color component.
D3DFVF_LASTBETA_UBYTE4	The description goes here
D3DFVF_NORMAL	Vertex format includes a vertex normal vector. This flag cannot be used with the D3DFVF_XYZRHW flag.
D3DFVF_PSIZE	Vertex format specified in point size. This size is expressed in camera space units for vertices that are not transformed and lit, and in device-space units for

	transformed and lit vertices.
D3DFVF_SPECULAR	Vertex format includes a specular color component.
D3DFVF_TEX0 - D3DFVF_TEX8	Number of texture coordinate sets for this vertex.
D3DFVF_XYZ	Vertex format includes the position of an untransformed vertex. This flag cannot be used with the D3DFVF_XYZRHW flag.
D3DFVF_XYZB1 - D3DFVF_XYZB5	Vertex format contains position data, and a corresponding number of weighting (beta) values to use for multimatrix vertex blending operations.
D3DFVF_XYZRHW	Vertex format contains transformed and clipped (x, y, z, w) data.

15.5.6 Light Types

Constants used with the [C3DLight::Create method](#)

Constant	Description
LIGHT_DIRECTIONAL	Creates a directional light.
LIGHT_POINT	Creates a point light.
LIGHT_SPOT	Creates a spot light.

15.5.7 Mesh Fill Styles

Constants used with the [C3DMesh::SetFill](#) method

Constant	Description
FILL_POINT	Fill points.
FILL_SOLID	Fill solid.
FILL_WIREFRAME	Fill wireframe, connecting each vertex with a line

15.5.8 Mesh Shading Styles

Constants used with the [C3DMesh::SetShading](#) method.

Constant	Description
SHADE_FLAT	Flat shading mode. The color and specular component of the first vertex in the triangle are used to determine the color and specular component of the face.
SHADE_GOURAUD	Gouraud shading mode. The color and specular components of the face are determined by a linear interpolation between all three of the triangle's vertices.

Additional Info:

The first vertex of a triangle for flat shading mode is defined in the following manner.

- For a triangle list, the first vertex of the triangle i is $i * 3$.
- For a triangle strip, the first vertex of the triangle i is vertex i .
- For a triangle fan, the first vertex of the triangle i is vertex $i + 1$.

15.5.9 Primitive Types

Constants used with the [C3DMesh::CreateMeshEx](#) method

Constant	Description
D3DPT_LINELIST	Renders the vertices as a list of isolated straight line segments.
D3DPT_LINESTRIP	Renders the vertices as a single polyline.
D3DPT_POINTLIST	Renders the vertices as a collection of isolated points. This value is unsupported for indexed primitives.
D3DPT_TRIANGLEFAN	Renders the vertices as a triangle fan.
D3DPT_TRIANGLELIST	Renders the specified vertices as a sequence of isolated triangles. Each group of three vertices defines a separate triangle. Back-face culling is affected by the current winding-order render state.
D3DPT_TRIANGLESTRIP	Renders the vertices as a triangle strip. The backface-culling flag is automatically flipped on even-numbered triangles.

15.5.10 Texture Blending Constants

Constants used by [C3DMesh::SetAlphaOperation](#) and [C3DMesh::SetColorOperation](#)

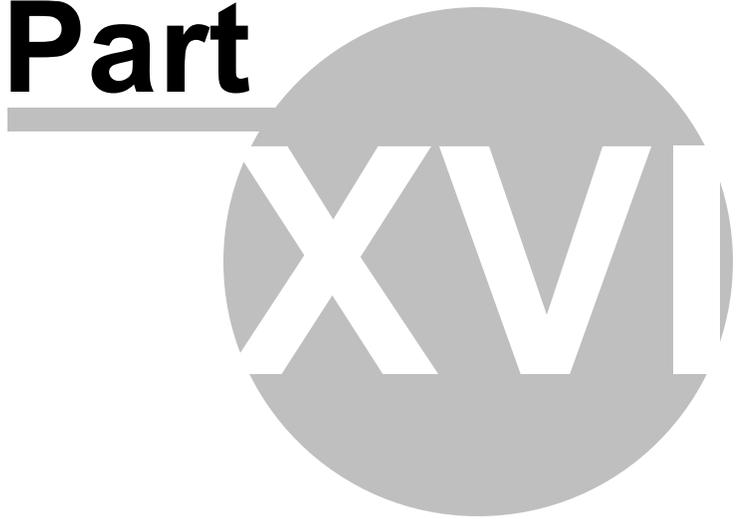
Constant	Description
D3DTOP_DISABLE	Disables output from this texture stage and all stages with a higher index. To disable texture mapping, set this as the color operation for the first texture stage (stage 0). Alpha operations cannot be disabled when color operations are enabled. Setting the alpha operation to D3DTOP_DISABLE when color blending is enabled causes undefined behavior.
D3DTOP_SELECTARG1	Use this texture stage's first color or alpha argument, unmodified, as the output.
D3DTOP_SELECTARG2	Use this texture stage's second color or alpha argument, unmodified, as the output.
D3DTOP_MODULATE	Multiply the components of the arguments.
D3DTOP_MODULATE2X	Multiply the components of the arguments, and shift the products to the left 1 bit (effectively multiplying them by 2) for brightening.
D3DTOP_MODULATE4X	Multiply the components of the arguments, and shift the products to the left 2 bits (effectively multiplying them by 4)

	for brightening.
D3DTOP_ADD	Add the components of the arguments.
D3DTOP_ADDSIGNED	Add the components of the arguments with a -0.5 bias, making the effective range of values from -0.5 through 0.5 .
D3DTOP_ADDSIGNED2X	Add the components of the arguments with a -0.5 bias, and shift the products to the left 1 bit.
D3DTOP_SUBTRACT	Subtract the components of the second argument from those of the first argument.
D3DTOP_ADDSMOOTH	Add the first and second arguments; then subtract their product from the sum.
D3DTOP_BLENDDIFFUSEALPHA	Linearly blend this texture stage, using the interpolated alpha from each vertex.
D3DTOP_BLENDTEXTUREALPHA	Linearly blend this texture stage, using the alpha from this stage's texture.
D3DTOP_BLENDFACTORALPHA	Linearly blend this texture stage.
D3DTOP_BLENDTEXTUREALPHA_PM	Linearly blend a texture stage that uses a premultiplied alpha.
D3DTOP_BLENDCURRENTALPHA	Linearly blend this texture stage, using the alpha taken from the previous texture stage.
D3DTOP_PREMODULATE	D3DTOP_PREMODULATE is set in stage n . The output of stage n is $arg1$.
D3DTOP_MODULATEALPHA_ADDCOLOR	Modulate the color of the second argument, using the alpha of the first argument; then add the result to argument one. This operation is supported only for color operations
D3DTOP_MODULATECOLOR_ADDALPHA	Modulate the arguments; then add the alpha of the first argument. This operation is supported only for color operations
D3DTOP_MODULATEINVALPHA_ADDCOLOR	Similar to D3DTOP_MODULATEALPHA_ADDCOLOR, but use the inverse of the alpha of the first argument. This operation is supported only for color operations
D3DTOP_MODULATEINVCOLOR_ADDALPHA	Similar to D3DTOP_MODULATECOLOR_ADDALPHA, but use the inverse of the color of the first argument. This operation is supported only for color operations
D3DTOP_BUMPENVMAP	Perform per-pixel bump mapping, using the environment map in the next texture stage, without luminance. This operation is supported only for color operations
D3DTOP_BUMPENVMAP_LUMINANCE	Perform per-pixel bump mapping, using the environment map in the next texture stage, with luminance. This operation is supported only for color operations
D3DTOP_DOTPRODUCT3	Modulate the components of each argument as signed components, add their products; then replicate the sum to all color channels, including alpha. This operation is supported for color and alpha operations.

D3DTOP_MULTIPLYADD	Performs a multiply-accumulate operation. It takes the last two arguments, multiplies them together, and adds them to the remaining input/source argument, and places that into the result.
D3DTOP_LERP	Linearly interpolates between the 2nd and 3rd source arguments by a proportion specified in the 1st source argument.

Database Programming Guide

Part



16 Database Programming Guide

16.1 Introduction

The Easy Database command set is a library included with the IWBASIC development environment. Allowing connection to any database format with an ODBC driver, the command set provides a robust solution for enabling SQL access to any application.

Main Features

- Supports accessing a database directly or by using a system DSN.
- Integrates with all IWBASIC variable types.
- Execute SQL statements directly or prepared.
- Easy handling of date, time and timestamp fields.
- Supports dynamic column and parameter binding.
- Can use the ODBC API directly with provided IWBASIC include files.
- Quickly iterate column, table and driver names with built in schema commands.
- Intuitive error handling.
- Create Microsoft Access compatible databases with one easy command.

Minimum Requirements

- IWBASIC 2.0 or greater
- ODBC 3.0 or greater installed. For Windows 2000 and XP it is already included on the system. For Windows 95, 98 and ME you can download the latest Microsoft Data Access Components from <http://msdn.microsoft.com/data/Default.aspx>. ODBC is also included with Microsoft Office.

16.2 Connecting to the database

The easy database command set interfaces with the ODBC API to perform all operations on a database. Communication with the database is done through the use of a driver supplied either with the Microsoft Data Access Components (MDAC) or by the database provider. MDAC includes popular database drivers for common formats such as Access and Paradox. .

Before any SQL queries, updates, insertions, etc. can be performed on a database you must first establish a connection to it. The connection is created using either the [dbConnect](#) connect function or the [dbConnectDSN](#) function.

A list of database drivers currently installed can be obtained by the [dbEnumDrivers](#) function.

Example:

```
DEF pDrivers as POINTER
pDrivers = dbEnumDrivers()

IF pDrivers <> NULL
    PRINT "Available database drivers:"
    FOR temp = EACH pDrivers as STRING
        PRINT #temp
    NEXT
    ListRemoveAll(pDrivers, TRUE)
ELSE
    PRINT "Unable to enumerate drivers"
ENDIF
```

Using dbConnect

dbConnect creates a connection to a database by supply the name of the driver to use, a filename if any, and any options. dbConnect returns a POINTER type that will be NULL if the connection could not be established. The return value must be saved in a variable of type POINTER to be passed to other commands and to disconnect from the database when operations are completed.

Example:

```
pdb = dbConnect("Microsoft Access Driver (*.mdb)", GETSTARTPATH + "inventory.mdb", "")
IF pdb <> NULL
    PRINT "Connection Established"
    dbDisconnect(pdb)
ENDIF
```

It is important to note that not every database type is discreet file based. In cases where a database is stored in a directory such as text format CSV files you supply the path to the directory in the filename parameter.

Example:

```
pdb = dbConnect("Microsoft Text Driver (*.txt; *.csv)", "c:\\csvdata\\", "")
```

And some drivers just use the options parameter and a keyword to connect. Consult the documentation for your database.

Using dbConnectDSN

Data Source Names (DSN) is the traditional method for connecting to a database through ODBC. While it may be traditional it is not the most convenient method unless you have a number of applications that will be using a database and don't want to worry about where the database is located. A DSN is created using the "Data Sources" control panel applet. After a DSN is created, and points to a valid database, connection to it is as easy as supplying the name and options.

Example.

```
pdb = dbConnectDSN("XYZ CORP", "UID=President;PWD=Private")
```

```
IF pdb <> NULL
  PRINT "Connection Established"
  dbDisconnect (pdb)
ENDIF
```

Disconnecting from the database

Your program must use [dbDisconnect](#) on the database before exiting. Simply pass the pointer returned by dbConnect or dbConnectDSN.

16.3 Catalog Functions

There are several system tables in a relational database which record information about tables, columns, etc. We call these system tables catalogs. When you want to know the schema of tables and indexes in a database, you can look at the catalogs to get the information.

Using the catalog functions you can create a view of the database dynamically without having to know ahead of time what tables are present and what columns are present in the tables.

Retrieving table names

To retrieve a list of tables contained in the database use the [dbListTables](#) function. A successful connection to the database must be established first using dbConnect or dbConnectDSN.

Example:

```
DEF pdb,pTables as POINTER
pdb = dbConnectDSN("XYZ CORP","")
IF pdb <> NULL    pTables = dbListTables (pdb)
  PRINT "Tables:"
  IF pTables <> NULL
    FOR temp = EACH pTables as STRING
      PRINT #temp
    NEXT
    ListRemoveAll (pTables,TRUE)
  ENDIF
  dbDisconnect (pdb)
ENDIF
```

As indicated in the above example dbListTables returns a pointer to an IWBASIC linked list. The list contains the names of all of the tables in the database as STRING types. After you are done with the list remember to delete it with ListRemoveAll(*listvar*, TRUE).

Retrieving column names

The names of the columns contained in a table, or returned in a result set, can be obtained by using [dbListColumns](#). A successful connection to the database must be established first using

dbConnect or dbConnectDSN.

Examples:

```
pColumns = dbListColumns(pdb, "", hstmt)
IF pColumns <> NULL
    FOR temp2 = EACH pColumns as STRING
        PRINT #temp2, ", ",
    NEXT
    PRINT
    ListRemoveAll(pColumns, TRUE)
ENDIF

pColumns = dbListColumns(pdb, "USERS")
IF pColumns <> NULL
    FOR temp2 = EACH pColumns as STRING
        PRINT #temp2, ", ",
    NEXT
    PRINT
    ListRemoveAll(pColumns, TRUE)
ENDIF
```

As indicated in the above example dbListColumns returns a pointer to an IWBasic linked list. After you are done with the list remember to delete it with ListRemoveAll(*listvar*, TRUE).

Getting a count of rows

The count of rows in a table, known as its cardinality, can be determined with the [dbCardinality](#) function.

Example:

```
DEF pdb, pTables as POINTER
pdb = dbConnectDSN("XYZ CORP", "")
IF pdb <> NULL
    pTables = dbListTables(pdb)
    PRINT "Tables:"
    IF pTables <> NULL
        FOR temp = EACH pTables as STRING
            PRINT #temp
            PRINT "Cardinality:", dbCardinality(pdb, #temp)
        NEXT
        ListRemoveAll(pTables, TRUE)
    ENDIF
    dbDisconnect(pdb)
ENDIF
```

16.4 SQL

Executing Queries

Structured Query Language (SQL) is the industry standard query language used for defining, organizing, managing, and retrieving data stored in relational databases. Unlike traditional procedural programming languages such as C and Pascal, you do not need to explicitly define how to perform a database operation. You can simply enter a request to the database using the English-

like SQL syntax, and the database will determine the best method to process the request and return the results to you when it is finished.

To execute any SQL statement use the [dbExecSQL](#) command or the [dbPrepareSQL](#) command. This section of the users guide will concentrate solely on constructing and executing statements. Retrieving results from an executed statement will be covered in the next section.

The functions provided by SQL go beyond simple data retrieval, although that is still one of its most important functions. SQL is actually divided into three parts, known as Data Definition Language (DDL), Data Manipulation Language (DML), and Data Control Language (DCL). Each of these performs a specific role, and together you can use them to perform all functions a DBMS provides, including:

- *Data definition* - lets you define the structure and organization of data and the relationships between data.
- *Data manipulation* -allows you to retrieve existing data from the database and update the database by adding new data, deleting old data, and modifying previously stored data.
- *Data control* -allows you to protect data against unauthorized access, and define integrity constraints to protect data from corruption.

Data Definition Language

The schema of a database is handled by a set of SQL statements in the SQL Data Definition Language (DDL). DDL makes use of the CREATE, DROP or ALTER SQL commands to define, remove or modify the definition of a database object. We will briefly explain the CREATE TABLE statement.

A database can contain many tables, and each table in a database stores information. Tables are composed of rows (records) and columns (fields). You can use the CREATE TABLE statement to create a new table in a database. The basic syntax of the SQL CREATE TABLE statement is:

CREATE TABLE table-name (column-name data-type,)

Example:

```
pdb = dbConnect("Microsoft Access Driver (*.mdb)", GETSTARTPATH+"db1.mdb","")
hstmt = dbExecSQL(pdb,"CREATE TABLE account (lname CHAR(15),fname CHAR(10), branch in
```

The ANSI/ISO SQL standard specifies a minimal set of data types that a DBMS should support. Almost all commercial SQL drivers support these data types. The table below lists the SQL data types and their IWBASIC equivalents.

Data Type	IWBASIC Equivalent	Description
CHAR(len)	STRING or ISTRING	Fixed-length character string
VARCHAR(len)	STRING or	Variable-length character string

	ISTRING	
BINARY(len)	Array/POINTER/ UDT	Binary data
COUNTER [(int,int)]	INT/UINT	Auto-increment integer. Optional start, amount
SMALLINT	WORD/SWORD	Small integer number
INTEGER [INT]	INT/UINT	Integer number
FLOAT	FLOAT	Low-precision floating point number
DOUBLE	DOUBLE	High-precision floating point number
DATE	DBDATE*	Date UDT
TIME	DBTIME*	Time UDT
TIMESTAMP	DBTIMESTAMP*	Timestamp UDT
DECIMAL [DEC] DECIMAL (precision,scale)	DOUBLE / STRING	Decimal numbers (use default precision and scale) Default precision is 17 and default scale is 6

*UDT's defined by the database command pak.

COUNTER is the keyword used by the Microsoft Access Driver for an auto-incrementing integer. Other drivers may use IDENTITY or some other keyword. Consult your database documentation for details.

ODBC handles cross type conversions. For example A string variable can be bound to a field containing numeric data and the driver will perform the conversion during retrieval.

Data Manipulation Language (DML)

Retrieving or manipulating the data in a database is handled by a set of SQL statements called the SQL Data Manipulation Language, or DML. The basic DML statements are SELECT, INSERT, DELETE and UPDATE.

Selecting Records

The basic syntax of the SELECT statement is:

SELECT item_list FROM table_list WHERE search_condition

The basic SELECT statement is made up of three components: SELECT, FROM and WHERE. The functions of each of these components is listed below:

- SELECT - specifies the columns or calculated columns to be retrieved by the query.
- FROM - specifies the tables that contain the items in the SELECT list.
- WHERE - specifies the search condition that must be met to select a row.

The WHERE clause may contain multiple search conditions. The search conditions that can be included in the WHERE clause are shown in the following list. They can include:

- Comparison operators (=, >, <, >=, <=, <>, !=)
- Ranges (BETWEEN and NOT BETWEEN)
- Lists (IN and NOT IN)
- String matches (LIKE and NOT LIKE)
- BLOB matches (MATCH and NOT MATCH)
- Unknown values (IS NULL and IS NOT NULL)
- Logical combinations (AND, OR)
- Negations (NOT)

For example, to perform a query to find all customers whose account balance is greater than \$10,000, you would use the following select statement:

```
hstmt = dbExecSQL(pdb,"SELECT lname, fname, balance FROM account WHERE balance > 10000
```

Inserting Records

The INSERT statement is used to add a new row to a table. The basic syntax of the INSERT statement is:

INSERT INTO table_name(column_names) VALUES value_list

The INSERT statement is made up of two components: INSERT INTO and VALUES. The function of these components is listed below:

- INSERT INTO - specifies the table you want to insert a row into. It can optionally contain a column list to specify that data should only be inserted into those columns. Columns not in this list will be inserted with NULL values.
- VALUES - specifies the data value you want to insert. You can insert values by using constants or parameters.

As stated above, the value list may contain constants or parameters. A constant is any numeric, text or date value that can be expressed in text form, such as 'John', 'Monday', 123, 54.823, etc. An example of the INSERT command using constants is shown below. This example adds a new account for John Smith to the database:

```
hstmt = dbExecSQL(pdb,"INSERT INTO account (lname, fname, branch, balance) VALUES ('j
```

Parameter data is represented by a question mark (?) in the value list, and values can be inserted later. Parameters can be used when the data values are unknown at preparation time, or when you want to save preparation time. An example of the INSERT command using parameters is shown below. This example is used to insert rows into the database, but the values are not currently known. The actual values to be inserted are bound before execution with the [dbBindParameter](#) command. After the statement is prepared it is executed with [dbExecute](#).

```
hstmt = dbPrepareSQL(pdb,"INSERT INTO account (lname, fname, branch) VALUES (?, ?, ?)")
```

Deleting Records

The DELETE statement deletes one or more rows from a table. The basic syntax of the DELETE statement is:

```
DELETE FROM table_name WHERE search_condition
```

The DELETE statement is made up of two components: DELETE FROM and WHERE. The function of these components is listed below:

- DELETE FROM - specifies the table you want to delete rows from.
- WHERE - specifies the search conditions that must be met to delete a row.

The WHERE clause may contain multiple search conditions. For a list of search conditions that can be included in the WHERE clause, refer to "Retrieving Data from the Database".

The following example will delete the account for John Smith from the database:

```
hstmt = dbExecSQL(pdb,"DELETE FROM account WHERE fname = 'john' AND lname = 'smith'")
```

Updating the Data

The UPDATE statement changes data in existing rows in a table. The basic syntax of the UPDATE statement is:

```
UPDATE table_name SET column_names expression WHERE search_condition
```

The UPDATE statement is made up of three components: UPDATE, SET and WHERE. The function of these components is listed below:

- UPDATE - specifies the table you want to update rows in.
- SET - specifies the columns you want to change and an expression that defines the changes to be made for each column.
- WHERE - specifies the search conditions that must be met to update a row.

The WHERE clause may contain multiple search conditions. For a list of search conditions that can be included in the WHERE clause, see the SQL Command and Function Reference.

The following example will add 6% interest to all accounts with a balance greater than \$1000.

```
hstmt = dbExecSQL(pdb,"UPDATE account SET balance = balance * 1.06 WHERE balance > 1000")
```

16.5 Retrieving Results

In the last section we covered executing SQL queries to add, delete, update and select records

from a database. Performing queries to retrieve data is one of the most important functions of a database.

Both [dbExecSQL](#) and [dbPrepareSQL](#) return a statement handle that is used to retrieve the data from selected rows. The entire set of rows returned by a query is called the *result set*. The statement handle must always be freed when you are done using it with the [dbFreeSQL](#) command.

Binding variables

Suppose we want to get last name, first name and branch information for customers at branch 11240 from the **account** table in a hypothetical banks database. To do this we might perform a query like the following:

```
SELECT lname, fname, branch FROM account WHERE branch = 11240
```

After preparing and executing this query, we are ready to fetch the data row by row. If all the information in the result columns in the projection of this query are known (e.g. data type, precision, etc.), then we can use [dbBindVariable](#) and [dbGet](#) to fetch the results.

Binding a variable associates it with a column in the result set. A bound variable is then updated with each call to the `dbGet`, `dbGetNext` or `dbGetPrev` functions. When binding variables it is important to remember that column numbers start at 1. Since our query has three column names after the `SELECT` statement there will be exactly three columns in the result set.

Example:

```
...
DEF last,first as STRING
DEF branch as INT
hstmt = dbExecSQL(pdb,"SELECT lname, fname, branch FROM account WHERE branch = 11240")
error = dbGetErrorCode(hstmt)
IF LEN(error)
    PRINT
    PRINT "Error Code: ", error
    PRINT "Error Text: ", dbGetErrorText(hstmt)
    PRINT
ELSE
    dbBindVariable(hstmt, 1, last)
    dbBindVariable(hstmt, 2, first)
    dbBindVariable(hstmt, 3, branch)
    WHILE dbGet(hstmt)
        PRINT last, " ", first
        PRINT branch
    ENDWHILE
ENDIF
dbFreeSQL(hstmt)
```

Direct data retrieval

Binding variables is the most common method for retrieving the data from a column. Some

database drivers also support retrieving the data directly without the need for binding. After a statement is successfully executed you can use [dbGetData](#), [dbGetTime](#), [dbGetDate](#), and [dbGetTimeStamp](#) to transfer the data from a result set column to a variable after each iteration of `dbGet`.

Binding variables will always result in more efficient database usage over larger result sets. Also certain drivers will not allow directly retrieving data from a column if it is already bound to a variable. The Access driver supports both binding and direct retrieval.

Example:

```
...
DEF last,first as STRING
DEF branch as INT
hstmt = dbExecSQL(pdb,"SELECT lname, fname, branch FROM account WHERE branch = 11240")
error = dbGetErrorCode(hstmt)
IF LEN(error)
    PRINT
    PRINT "Error Code: ", error
    PRINT "Error Text: ", dbGetErrorText(hstmt)
    PRINT
ELSE
    WHILE dbGet(hstmt)
        dbGetData(hstmt, 1, last)
        dbGetData(hstmt, 2, first)
        dbGetData(hstmt, 3, branch)
        PRINT last, " ", first
        PRINT branch
    ENDWHILE
ENDIF
dbFreeSQL(hstmt)
```

Dealing with NULL columns

When thinking of the term 'NULL' most programmers envision a NULL pointer, or a NULL string, or the number 0. A database column in a result set can follow this methodology for any data type.

A NULL column in a result set is simply a column that contains no data, there is nothing to read. For obvious reasons the database can't use the number 0 to represent a NULL value so instead a field is tagged with a constant. The database command set provides two methods for determining whether a column is currently NULL.

The first method involves supplying an optional indicator variable to `dbBindVariable` that on each success call to any of the `dbGet` functions will be used by the database driver to store a value. This value is either the length of the returned data or one of the constants `SQL_NULL_DATA` or `SQL_NO_TOTAL`. The one we are interested in is `SQL_NULL_DATA` which happens to be equal to -1

Example:

```
DEF last,first as STRING
DEF branch,branch_null as INT
```

```

hstmt = dbExecSQL(pdb,"SELECT lname, fname, branch FROM account WHERE branch = 11240")
error = dbGetErrorCode(hstmt)
IF LEN(error)
  PRINT
  PRINT "Error Code: ", error
  PRINT "Error Text: ", dbGetErrorText(hstmt)
  PRINT
ELSE
  dbBindVariable(hstmt, 1, last)
  dbBindVariable(hstmt, 2, first)
  dbBindVariable(hstmt, 3, branch, branch_null)
  WHILE dbGet(hstmt)
    PRINT last, " ", first
    IF branch_null <> -1
      PRINT branch
    ELSE
      PRINT "No longer an account holder"
    ENDIF
  ENDWHILE
ENDIF
dbFreeSQL(hstmt)

```

The second method to determine if a column is NULL is to use the [dbIsNull](#) function. `dbIsNull` returns TRUE if a column in a result set is currently NULL or FALSE otherwise. `dbIsNull` will not work with all database drivers and others will not allow a bound column to be checked against NULL in this manner. The Access driver allows both.

Example:

```

DEF last,first as STRING
DEF branch as INT
hstmt = dbExecSQL(pdb,"SELECT lname, fname, branch FROM account WHERE branch = 11240")
error = dbGetErrorCode(hstmt)
IF LEN(error)
  PRINT
  PRINT "Error Code: ", error
  PRINT "Error Text: ", dbGetErrorText(hstmt)
  PRINT
ELSE
  dbBindVariable(hstmt, 1, last)
  dbBindVariable(hstmt, 2, first)
  dbBindVariable(hstmt, 3, branch)
  WHILE dbGet(hstmt)
    PRINT last, " ", first
    IF dbIsNull(hstmt,3) = FALSE
      PRINT branch
    ELSE
      PRINT "No longer an account holder"
    ENDIF
  ENDWHILE
ENDIF
dbFreeSQL(hstmt)

```

Which method you chose depends on the needs of the program and the capabilities of the database driver. A final method not outlined here is to adjust the SQL query to not return rows based on a NULL criteria such as:

```
hstmt = dbExecSQL(pdb,"SELECT lname, fname, branch FROM account WHERE branch IS NOT  
NULL")
```

16.6 Updates and Insertions

Updating and inserting new records (rows) into a table is easily accomplished using SQL. Data can be specified in the SQL statement as constants or parameterized by binding variables.

Using SQL constants

If updates and insertions are done infrequently on the database it is convenient to just construct the SQL statement on the fly inserting values as constants and then calling dbExecSQL to perform the action.

Example:

```
REM val$ can be constructed elsewhere  
val$ = "'Mulder','Fox', 11240, 10011232, 3000.00"  
hstmt = dbExecSQL(pdb,"INSERT INTO accounts VALUES(" + val$ + ")")
```

The INSERT statement can accept an optional column list if not all values are known. Any column names not in the list are filled with NULL values. There must be a one to one matching of column names and values

```
val$ = "'Mulder','Fox', 11240, 10011232"  
hstmt = dbExecSQL(pdb,"INSERT INTO accounts (lname, fname, branch, acctno) VALUES(" +
```

Updates of existing rows is just as simple. The UPDATE statement allows specifying the columns to update and the a search condition to match. Updating a single row is then accomplished by using a unique key. In our example of a fictional banks database the account number is a uniquely identifying column.

Example:

```
val$ = "3000.00"  
acctno$ = "10011232"  
hstmt = dbExecSQL(pdb,"UPDATE accounts SET balance = " + val$ + " WHERE acctno = " + acctno$)
```

Using parameters

Looking at the examples for using SQL constants you can see that its not a very efficient way of updating and inserting data. A better method for a large number of operations is to used bound parameters.

A parameter is used in a SQL statement when:

- values of the parameters are unknown at preparation time
- applications need to execute the same SQL statment several times with different parameter values
- applications need to convert the parameter values between different data types

For example, an application wants to insert five rows into a table named accounts.

```
INSERT INTO accounts (lname, fname, branch, acctno) VALUES (?, ?, ?, ?)
```

In this statement, ? is the parameter marker. By using parameters, the application only needs to prepare this statement once, and then execute the prepared statement five times with different parameter values to insert five rows into the account table.

Using parameters requires a two step execution of the SQL statement, and binding the necessary variables to be used as parameters.

Step 1. Prepare the SQL statement

[dbPrepareSQL](#) prepares an SQL statement for later execution, in this manner it can be executed many times using the same statement handle.

```
hstmt = dbPrepareSQL(pdb, "INSERT INTO accounts (lname, fname, branch, acctno) VALUES
```

Step 2. Bind parameters

[dbBindParameter](#) associates a variable with a parameter in the prepared statement. Parameter numbers are one based

```
DEF last, first as STRING
DEF branch, acctno as INT
dbBindParameter(hstmt, 1, last, 255)
dbBindParameter(hstmt, 2, first, 255)
dbBindParameter(hstmt, 3, branch)
dbBindParameter(hstmt, 4, acctno)
```

Step 3. fill in the variables and execute the statement one or more times.

[dbExecute](#) executes the prepared SQL statement, substituting any bound parameters as needed.

```
last = "Thompson"
first = "Tom"
branch = 11223
acctno = 11014457
dbExecute(hstmt)

last = "The Grey"
first = "Gandalf"
branch = 11223
acctno = 110144566
dbExecute(hstmt)
```

With all statement handles remember to use [dbFreeSQL](#) when you are finished. To bind a DBDATE, DBTIME or DBTIMESTAMP UDT parameter use the [dbBindDateParam](#),

[dbBindTimeParam](#) and [dbBindTimeStampParam](#) commands instead of dbPindParameter.

16.7 Accessing the ODBC API

The necessary include files for directly accessing the ODBC API are provided for advanced programmers with more complex database needs. These include files are installed to your IW BASIC include directory and are named 'sql.inc', 'sqltypes.inc' and 'sqlxext.inc'. These files define all of the constants and functions used by the API.

It is only necessary to include the main sqlxext.inc file with **\$INCLUDE "sqlxext.inc"** to begin using the API. The other files are brought in automatically. The files are intended for use with ODBC version 3.0 or higher. To define constants introduced in ODBC version 3.5 add

```
$define ODBCVER35
```

before the \$INCLUDE statement for any file that may need those constants.

Here is a short example that uses the ODBC API directly and shows the basic steps involved in connecting to a database:

```
$INCLUDE "sqlxext.inc"

DEF hEnv,hDBC as UINT
DEF hStmt as UINT
DEF rc,cbOut as SWORD
DEF strConnect[1023],strOut[1023] as ISTRING

rc = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, hEnv)
IF(rc = SQL_SUCCESS) OR (rc = SQL_SUCCESS_WITH_INFO)
  SQLSetEnvAttr(hEnv,SQL_ATTR_ODBC_VERSION,SQL_OV_ODBC3,SQL_IS_INTEGER)
  rc = SQLAllocHandle(SQL_HANDLE_DBC, hEnv, hDbc)
  IF(rc = SQL_SUCCESS) OR (rc = SQL_SUCCESS_WITH_INFO)
    'ODBC initialized try and connect to a database
    strConnect = "DSN=IW BASIC TEST"
    rc = SQLDriverConnect(hDbc,NULL,strConnect,LEN(strConnect),strOut,1023,cbOut,SQL_DRIVER_NOPROMPT)
    IF rc = SQL_SUCCESS
      'connection established and at this point
      'SQL can be executed, etc. But we will just report success
      'and exit
      PRINT "Database connection established"
      SQLDisconnect(hDbc)
      SQLFreeHandle(SQL_HANDLE_DBC,hDbc)
      SQLFreeHandle(SQL_HANDLE_ENV,hEnv)
    ELSE
      'error so free the connection handle and environment handle
      PRINT "Error connecting"
      SQLFreeHandle(SQL_HANDLE_DBC,hDbc)
      SQLFreeHandle(SQL_HANDLE_ENV,hEnv)
    ENDIF
  ELSE
    'error so free the environment handle
```

```

        PRINT "Error allocating connection handle"
        SQLFreeHandle(SQL_HANDLE_ENV,hEnv)
    ENDIF
ELSE
    PRINT "Error allocating environment handle"
ENDIF

PRINT "Any key to close"
DO:UNTIL INKEY$ <> ""
END

```

Combining API access with the command pak.

You can mix direct API calls with commands from the command pak. For example `dbGet`, `dbGetNext`, and `dbGetPrev` take only a statement handle allocated with `SQLAllocHandle`. To use commands that require a database pointer it is necessary to create a UDT that contains both the environment and connection handles. The UDT is defined as:

```

TYPE DBConnection
    DEF hEnv as UINT
    DEF hDbc as UINT
ENDTYPE

```

Create a variable of type `DBConnection` and fill in the members with the connection and environment handles returned by `SQLAllocHandle`. For example in the above code we use `dbExecSQL` after a connection is established with a few modifications:

```

DEF DBC as DBConnection
...
        'connection established and at this point
        DBC.hEnv = hEnv
        DBC.hDbc = hDbc
        hstmt = dbExecSQL(DBC, "SELECT * FROM addresses")

```

16.8 Alphabetical Command Reference

16.8.1 dbBindDate

Syntax

`INT = dbBindDate(hstmt as INT, column as INT, date as DBDATE, opt pReturn as POINTER)`

Description

Binds a `DBDATE` variable to a column in a result set.

Parameters

hstmt - A statement handle returned by `dbExecSQL` or `dbPrepareSQL`.

column - The ones based column number to bind.

date - A variable of type `DBDATE`.

pReturn - optional pointer to an INT.

Return value

TRUE if variables successfully bound to column. FALSE on error.

Remarks

The variable passed to the optional *pReturn* value will contain an indicator of the operation after each `dbGet`, `dbGetNext` or `dbGetPrev` operation. The variable can then be checked against `SQL_NULL_DATA` or `SQL_NO_TOTAL`. It is easier to use the `dbIsNull` function for checking for a NULL column. Some database drivers won't allow using `dbIsNull` against bound columns and this provides an alternative.

See Also: [dbBindVariable](#), [dbBindTime](#), [dbBindTimeStamp](#)

Example usage

```
DEF dtAdded as DBDATE
DEF first,last as STRING
DEF Address_ID as INT
...
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses")
IF hstmt
    'bind the columns in the table to appropriate variables
    'column numbers start at 1 starting from the left.

    dbBindVariable(hstmt,1,Address_ID)
    dbBindVariable(hstmt,2,first)
    dbBindVariable(hstmt,3,last)
    dbBindDate(hstmt,8,dtAdded)
    WHILE dbGet(hstmt)
        PRINT "ID:",Address_ID
        PRINT "Name:",first,last
        PRINT "Date added:",USING("## ## ####",dtAdded.month,dtAdded.day,dtAdded.year)
    ENDWHILE
ENDIF
```

16.8.2 dbBindDateParam

Syntax

`INT = dbBindDateParam(hstmt as INT,param as INT,dt as DBDATE)`

Description

Binds a DBDATE variable to be used as a parameter in a prepared SQL statement.

Parameters

hstmt - A statement handle returned by `dbPrepareSQL`.

param - The ones based parameter number to bind.

dt - A UDT of type DBDATE

Return value

TRUE if variable bound successfully, FALSE otherwise.

Remarks

A parameter is denoted by a ? in the SQL statement. There must be exactly one bound variable for each parameter.

Example usage

```

DEF bd as DBDATE
hstmt = dbPrepareSQL(pdb,"INSERT INTO Addresses (FirstName,LastName,Address,Birthday)
IF hstmt
    dbBindParameter(hstmt,1,first,255)
    dbBindParameter(hstmt,2,last,255)
    dbBindParameter(hstmt,3,street,255)
    dbBindDateParam(hstmt,4,bd)
    'after the variables are bound you can insert as many records as needed with one s
    first = "Lisa"
    last = "Jones"
    street = "123 Niagara"
    bd.day = 1:bd.month = 7:bd.year = 1959
    dbExecute(hstmt)
    '
    first = "Tammy"
    last = "Miller"
    street = "123 Blah Blah"
    bd.day = 25:bd.month = 12:bd.year = 1962
    dbExecute(hstmt)
    '
    dbFreeSQL(hstmt)
ENDIF

```

16.8.3 dbBindParameter**Syntax**

INT = dbBindParameter(hstmt as INT,param as INT,variable as ANYTYPE,opt cbSize as INT)

Description

Binds a variable to be used as a parameter in a prepared SQL statement.

Parameters

hstmt - A statement handle returned by dbPrepareSQL.

param - The ones based parameter number to bind.

variable - The variable to bind as a parameter.

cbSize - For string and character types.

Return value

TRUE if variable bound successfully, FALSE otherwise.

Remarks

For string types cbSize must be set to the maximum string length a column can hold. A parameter is denoted by a ? in the SQL statement. There must be exactly one bound variable for each parameter.

Example usage

```
hstmt = dbPrepareSQL(pdb,"INSERT INTO Addresses (FirstName,LastName,Address) VALUES (?,
IF hstmt
    dbBindParameter(hstmt,1,first,255)
    dbBindParameter(hstmt,2,last,255)
    dbBindParameter(hstmt,3,street,255)
    'after the variables are bound you can insert as many records as needed with one s
    first = "Lisa"
    last = "Jones"
    street = "123 Niagara"
    dbExecute(hstmt)
    '
    first = "Tammy"
    last = "Miller"
    street = "123 Blah Blah"
    dbExecute(hstmt)
    '
    dbFreeSQL(hstmt)
ENDIF
```

16.8.4 dbBindTime

Syntax

INT = dbBindTime(hstmt as INT,column as INT,time as DBTIME,opt pReturn as POINTER)

Description

Binds a DBTIME variable to a column in a result set.

Parameters

hstmt - A statement handle returned by dbExecSQL or dbPrepareSQL.

column - The ones based column number to bind.

time - A variable of type DBTIME.

pReturn - optional pointer to an INT.

Return value

TRUE if variable successfully bound to column. FALSE on error.

Remarks

The variable passed to the optional pReturn value will contain an indicator of the operation after each dbGet, dbGetNext or dbGetPrev operation. The variable can then be checked against SQL_NULL_DATA or SQL_NO_TOTAL. It is easier to use the dbIsNull function for checking for a NULL column. Some database drivers won't allow using dbIsNull against bound columns and this provides an alternative.

See Also: [dbBindVariable](#), [dbBindDate](#), [dbBindTimeStamp](#)

Example usage

```
DEF tmAdded as DBTIME
DEF first,last as STRING
DEF Address_ID as INT
```

```

...
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses")
IF hstmt
  'bind the columns in the table to appropriate variables
  'column numbers start at 1 starting from the left.
  '
  dbBindVariable(hstmt,1,Address_ID)
  dbBindVariable(hstmt,2,first)
  dbBindVariable(hstmt,3,last)
  dbBindDate(hstmt,9,tmAdded)
  WHILE dbGet(hstmt)
    PRINT "ID:",Address_ID
    PRINT "Name:",first,last
    PRINT "Time added:",USING("##:##:##",tmAdded.hour,tmAdded.minute,tmAdded.second)
  ENDWHILE
ENDIF

```

16.8.5 dbBindTimeParam

Syntax

INT = dbBindTimeParam(hstmt as INT,param as INT,tm as DBTIME)

Description

Binds a DBTIME variable to be used as a parameter in a prepared SQL statement.

Parameters

hstmt - A statement handle returned by dbPrepareSQL.

param - The ones based parameter number to bind.

tm - A UDT of type DBTIME

Return value

TRUE if variable bound successfully, FALSE otherwise.

Remarks

A parameter is denoted by a ? in the SQL statement. There must be exactly one bound variable for each parameter.

Example usage

```

DEF tm as DBTIME
hstmt = dbPrepareSQL(pdb,"INSERT INTO Addresses (FirstName,LastName,Address,tmAdded) VALUES (?, ?, ?, ?)")
IF hstmt
  dbBindParameter(hstmt,1,first,255)
  dbBindParameter(hstmt,2,last,255)
  dbBindParameter(hstmt,3,street,255)
  dbBindTimeParam(hstmt,4,tm)
  'after the variables are bound you can insert as many records as needed with one statement
  first = "Lisa"
  last = "Jones"
  street = "123 Niagara"
  tm.hour = 12:tm.minute = 0:tm.second = 0
  dbExecute(hstmt)
  '
  first = "Tammy"

```

```
last = "Miller"
street = "123 Blah Blah"
tm.hour = 9:tm.minute = 35:tm.second = 0
dbExecute(hstmt)
'
dbFreeSQL(hstmt)
ENDIF
```

16.8.6 dbBindTimeStamp

Syntax

INT = dbBindTimeStamp(hstmt as INT,column as INT,timestamp as DBTIMESTAMP,opt
pReturn as POINTER)

Description

Binds a DBTIMESTAMP variable to a column in a result set.

Parameters

hstmt - A statement handle returned by dbExecSQL or dbPrepareSQL.

column - The ones based column number to bind.

timestamp - A variable of type DBTIMESTAMP.

pReturn - optional pointer to an INT.

Return value

TRUE if variable successfully bound to column. FALSE on error.

Remarks

The variable passed to the optional pReturn value will contain an indicator of the operation after each dbGet, dbGetNext or dbGetPrev operation. The variable can then be checked against SQL_NULL_DATA or SQL_NO_TOTAL. It is easier to use the dbIsNull function for checking for a NULL column. Some database drivers won't allow using dbIsNull against bound columns and this provides an alternative.

See Also: [dbBindVariable](#), [dbBindDate](#), [dbBindTime](#)

Example usage

```
DEF tsAdded as DBTIMESTAMP
DEF first,last as STRING
DEF Address_ID as INT
...
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses")
IF hstmt
  'bind the columns in the table to appropriate variables
  'column numbers start at 1 starting from the left.
  '
  dbBindVariable(hstmt,1,Address_ID)
  dbBindVariable(hstmt,2,first)
  dbBindVariable(hstmt,3,last)
  dbBindTimeStamp(hstmt,8,tsAdded)
  WHILE dbGet(hstmt)
    PRINT "ID:",Address_ID
```

```

    PRINT "Name:", first, last
    PRINT "Time added:", USING("##:##:##", tsAdded.hour, tsAdded.minute, tsAdded.second)
    PRINT "Date added:", USING("## / ## / ####", tsAdded.month, tsAdded.day, tsAdded.year)
  ENDWHILE
ENDIF

```

16.8.7 dbBindTimeStampParam

Syntax

INT = dbBindTimeStampParam(hstmt as INT, param as INT, ts as DBTIMESTAMP)

Description

Binds a DBTIMESTAMP variable to be used as a parameter in a prepared SQL statement.

Parameters

hstmt - A statement handle returned by dbPrepareSQL.

param - The ones based parameter number to bind.

ts - A UDT of type DBTIMESTAMP

Return value

TRUE if variable bound successfully, FALSE otherwise.

Remarks

A parameter is denoted by a ? in the SQL statement. There must be exactly one bound variable for each parameter.

Example usage

```

DEF ts as DBTIMESTAMP
hstmt = dbPrepareSQL(pdb, "INSERT INTO Addresses (FirstName, LastName, Address, dtAdded) VALUES (?, ?, ?, ?)")
IF hstmt
  dbBindParameter(hstmt, 1, first, 255)
  dbBindParameter(hstmt, 2, last, 255)
  dbBindParameter(hstmt, 3, street, 255)
  dbBindTimeStampParam(hstmt, 4, ts)
  'after the variables are bound you can insert as many records as needed with one statement
  first = "Lisa"
  last = "Jones"
  street = "123 Niagara"
  ts.day=23:ts.month=9:ts.year=2004:ts.hour = 12:ts.minute = 0:ts.second = 0
  dbExecute(hstmt)
  '
  first = "Tammy"
  last = "Miller"
  street = "123 Blah Blah"
  ts.day=23:ts.month=9:ts.year=2004:ts.hour = 12:ts.minute = 0:ts.second = 0
  dbExecute(hstmt)
  '
  dbFreeSQL(hstmt)
ENDIF

```

16.8.8 dbBindVariable

Syntax

```
INT = dbBindVariable(hstmt as INT,column as INT,variable as ANYTYPE,opt pReturn as  
POINTER,opt cbSize as INT)
```

Description

Binds a variable to a column in a result set.

Parameters

hstmt - A statement handle returned by dbExecSQL.

column - The ones based column to bind.

variable - The variable that will receive the data from the bound column.

pReturn - Optional. Pointer to an INT.

cbSize - Optional. For strings and binary data specifies maximum length.

Return value

TRUE if variable successfully bound to column. FALSE on error.

Remarks

The variable passed to the optional pReturn value will contain an indicator of the operation after each dbGet, dbGetNext or dbGetPrev operation. The variable can then be checked against SQL_NULL_DATA or SQL_NO_TOTAL. It is easier to use the dbIsNull function for checking for a NULL column. Some database drivers won't allow using dbIsNull against bound columns and this provides an alternative.

Each bound variable will be filled in with the data contained in the record retrieved with dbGet, dbGetNext or dbGetPrev.

cbSize defaults to 255. Binary data can be retrieved by using a UDT and setting cbSize to the length of the UDT.

See Also: [dbBindTime](#), [dbBindDate](#), [dbBindTimeStamp](#)

Example usage

```
DEF first,last as STRING  
DEF Address_ID as INT  
...  
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses")  
IF hStmt  
    'bind the columns in the table to appropriate variables  
    'column numbers start at 1 starting from the left.  
  
    dbBindVariable(hstmt,1,Address_ID)  
    dbBindVariable(hstmt,2,first)  
    dbBindVariable(hstmt,3,last)  
    WHILE dbGet(hstmt)  
        PRINT "ID:",Address_ID  
        PRINT "Name:",first,last  
    ENDWHILE  
ENDIF
```

16.8.9 dbCardinality

Syntax

INT = dbCardinality(*pConnection* as POINTER,*tableName* as STRING)

Description

Returns the cardinality of a table. The count of the number of rows..

Parameters

pConnection - Database pointer returned by dbConnect or dbConnectDSN.

tableName - Name of the table to retrieve cardinality.

Return value

The count of all rows in the table or 0 if no rows exist..

Remarks

dbCardinality is an expensive function to use in respect to time and resources. It should be used sparingly with the result saved for future use.

Example usage

```
FOR temp = EACH pTables as STRING
  PRINT #temp
  PRINT "Cardinality:", dbCardinality (pdb, #temp)
NEXT
```

16.8.10 dbConnect

Syntax

POINTER = dbConnect(*driver* as STRING,*filename* as STRING,*options* as STRING,*OPT*
parwnd as POINTER)

Description

Connects to a database directly by specifying a driver name and options.

Parameters

driver - Name of database driver to use for connection.

filename - Filename for database

options - Parameters for connection

parwnd - Optional. IW BASIC WINDOW type as the parent for error dialogs.

Return value

A pointer to the connected database or NULL if the connection could not be established.

Remarks

The names of all ODBC drivers on the system can be obtained by the dbEnumDrivers function. Not all drivers are file based and the filename parameter can be left as an empty string "". The options are driver dependent.

A successful connection must be established before any SQL statements can be executed. Before the application exits it needs to disconnect from the driver using `dbDisconnect`

See Also: [dbEnumDrivers](#), [dbConnectDSN](#), [dbDisconnect](#)

Example usage

```
pdb = dbConnect("Microsoft Access Driver (*.mdb)", GETSTARTPATH + "db1.mdb", "UID=admin;
```

16.8.11 dbConnectDSN

Syntax

```
POINTER = dbConnectDSN(DSNname as STRING, options as STRING, OPT parwnd as POINTER)
```

Description

Connects to a database using an ODBC data source name (DSN).

Parameters

DSNname - Data source name as shown in the Data Sources (ODBC) control panel.

options - Connection options for the database

parwnd - Optional. IW BASIC WINDOW type to use as the parent for error dialogs.

Return value

A pointer to the connected database or NULL if the connection could not be established.

Remarks

The options a DSN supports is driver dependant. Any options supplied are given to the driver in addition to the ones specified in the DSN record. In most cases the string should be left as an empty string "".

A successful connection must be established before any SQL statements can be executed. Before the application exits it needs to disconnect from the driver using `dbDisconnect`

See Also: [dbEnumDrivers](#), [dbConnect](#), [dbDisconnect](#)

Example usage

```
pdb = dbConnectDSN("INVENTORY_XYZ", "")
```

16.8.12 dbCreateMDB

Syntax

```
INT = dbCreateMDB(path as STRING)
```

Description

Create a Microsoft Access compatible database file.

Parameters

path - Full path and filename for the database.

Return value

TRUE if database was successfully created, FALSE if there was an error or the database already exists.

Remarks

You can establish a connection the the database as soon as it is created.

Example usage

```
IF dbCreateMDB("c:\\myfiles\\addresses.mdb")
ENDIF
```

16.8.13 dbDisconnect**Syntax**

dbDisconnect(*pConnection* as POINTER)

Description

Disconnects from a database.

Parameters

pConnection - Pointer to database returned by dbConnect or dbConnectDSN.

Return value

None

Remarks

All statement handles must be freed with dbFreeSQL before you disconnect from the database.

See Also: [dbConnect](#), [dbConnectDSN](#)

Example usage

```
pdb = dbConnect("Microsoft Access Driver (*.mdb)", GETSTARTPATH + "db1.mdb", "")
IF pdb <> NULL
    PRINT "Connection established to " + GETSTARTPATH + "db1.mdb"
    dbDisconnect(pdb)
ENDIF
```

16.8.14 dbEnumDrivers**Syntax**

POINTER = dbEnumDrivers()

Description

Enumerates all ODBC database drivers available on the system.

Parameters

None.

Return value

A pointer to a linked list of database names or NULL if no drivers are currently installed.

Remarks

The pointer returned is a standard IW BASIC linked list type. It must be deleted with ListRemoveAll when you are finished enumerating the driver names. Each element of the list is a pointer to a string containing a database driver name.

Example usage

```
DEF pDrivers as POINTER
pDrivers = dbEnumDrivers()

IF pDrivers <> NULL
    PRINT "Available database drivers:"
    FOR temp = EACH pDrivers as STRING
        PRINT #temp
    NEXT
    ListRemoveAll(pDrivers, TRUE)
ELSE
    PRINT "Unable to enumerate drivers"
ENDIF
```

16.8.15 dbExecSQL**Syntax**

HANDLE = dbExecSQL(pConnection as POINTER, statement as STRING)

Description

Immediately executes an SQL statement for the database.

Parameters

pConnection - A pointer to a database returned by dbConnect or dbConnectDSN.

statement - The SQL statement to execute

Return value

A handle to the executed SQL statement or NULL if a statement handle could not be allocated.

Remarks

Use dbGetErrorCode or dbGetErrorText to determine whether the statement was successfully executed. The application can include one or more parameter markers in the SQL statement. To include a parameter marker, the application embeds a question mark (?) into the SQL statement at the appropriate position. When finished with the returned statement handle the application must free it with dbFreeSQL.

See Also: [dbFreeSQL](#), [dbPrepareSQL](#), [dbExecute](#), [dbFreeSQL](#)

Example usage

```
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses WHERE FirstName = 'John'")
error = dbGetErrorCode(hstmt)
IF LEN(error)
    PRINT
    PRINT "Error Code: ", error
    PRINT "Error Text: ", dbGetErrorText(hstmt)
    PRINT
ENDIF
```

16.8.16 dbExecute**Syntax**

INT = dbExecute(hstmt as INT)

Description

Executes a prepared statement, using the current values of the parameter marker variables if any parameter markers exist in the statement.

Parameters

hstmt - Statement handle returned by dbPrepareSQL.

Return value

TRUE if statement was successfully executed. FALSE if there was an error.

Remarks

Prepared statements are used to perform many duplicate updates or inserts into a database table. Using a prepared statement allows simply updating the bound parameters and calling dbExecute for each iteration. The main advantage over direct execution with dbExecSQL is speed of updates and insertions.

See Also: [dbPrepareSQL](#), [dbFreeSQL](#)

Example usage

```
hstmt = dbPrepareSQL(pdb,"INSERT INTO Addresses (FirstName,LastName,Address) VALUES(?, ?, ?)")
IF hstmt
    dbBindParameter(hstmt,1,first,255)
    dbBindParameter(hstmt,2,last,255)
    dbBindParameter(hstmt,3,street,255)
    'after the variables are bound you can insert as many records as needed with one s
    first = "Lisa"
    last = "Jones"
    street = "123 Niagara"
    dbExecute(hstmt)
    '
    first = "Tammy"
    last = "Miller"
    street = "123 America St"
    dbExecute(hstmt)
    '
dbFreeSQL(hstmt)
```

```
ENDIF
```

16.8.17 dbFreeSQL

Syntax

```
dbFreeSQL(hstmt as INT)
```

Description

Frees an SQL statement releasing any memory used.

Parameters

hstmt - Statement handle returned by dbExecSQL or dbPrepareSQL

Return value

None.

Remarks

All statement handles should be freed before calling dbDisconnect on the database pointer.

Example usage

```
dbFreeSQL(hstmt)
```

16.8.18 dbGet

Syntax

```
INT = dbGet(hstmt as INT)
```

Description

Gets the next record from a result set.

Parameters

hstmt - Statement handle returned by dbExecSQL or dbPrepareSQL.

Return value

TRUE if the record was retrieved and any bound variables filled in with column data. FALSE if the end of the result set was reached or there were no records in the result set.

Remarks

A column that is nullable and currently set to NULL can be determined by dbIsNull after a dbGet operation. A NULL value can also be determined by checking the optional 'pReturn' variable used when binding the column with dbBindVariable.

See Also: [dbGetNext](#), [dbGetPrev](#), [dbExecSQL](#), [dbPrepareSQL](#)

Example usage

```
hstmt = dbExecSQL(pdb, "SELECT * FROM Addresses")
error = dbGetErrorCode(hstmt)
IF LEN(error)
```

```

PRINT
PRINT "Error Code: ", error
PRINT "Error Text: ", dbGetErrorText(hstmt)
PRINT
ENDIF

IF hstmt
  'bind the columns in the table to appropriate variables
  'column numbers start at 1 starting from the left.
  dbBindVariable(hstmt,1,Address_ID)
  dbBindVariable(hstmt,2,first)
  dbBindVariable(hstmt,3,last)
  dbBindVariable(hstmt,4,street)
  dbBindVariable(hstmt,5,city)
  dbBindVariable(hstmt,6,state)
  dbBindVariable(hstmt,7,zip)
  'dbGet returns TRUE if data has been retrieved and stored
  'in the bound variables. Or FALSE if the end of data has been reached or an error
  WHILE dbGet(hstmt)
    PRINT "ID:",Address_ID
    PRINT first," ",last
    PRINT street
    IF dbIsNull(hstmt,5) = FALSE
      PRINT city, " ",state, " ", zip
    ENDIF
    PRINT
  ENDWHILE
  dbFreeSQL(hstmt)
ENDIF

```

16.8.19 dbGetData

Syntax

INT = dbGetData(hstmt as INT,column as INT,variable as ANYTYPE,opt cbSize as INT)

Description

Gets data directly from a column after a dbGet, dbGetNext or dbGetPrev operation.

Parameters

hstmt - Statement handle returned by dbExecSQL or dbPrepareSQL.

column - Ones based column number to retrieve data from.

variable - Variable to store retrieved data into.

cbSize - Optional. For string and raw binary data you must specify the maximum length. Defaults to 255.

Return value

TRUE if data was successfully retrieved and stored in the variable. FALSE on error or empty result set.

Remarks

dbGetData is an alternative to binding variables and is not supported by all database drivers. Certain drivers will not allow directly retrieving data from a column if it is already bound to a

variable. The Access driver supports both binding and direct retrieval. Binding variables will always result in faster retrieval of data.

dbGetData supports most of the IW BASIC built in types and raw binary data.

See Also: [dbGetTime](#), [dbGetDate](#), [dbGetTimeStamp](#)

Example usage

```
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses")
error = dbGetErrorCode(hstmt)
IF LEN(error)
  PRINT
  PRINT "Error Code: ", error
  PRINT "Error Text: ", dbGetErrorText(hstmt)
  PRINT
ENDIF

IF hstmt
  WHILE dbGet(hstmt)
    dbGetData(hstmt,1,Address_ID)
    dbGetData(hstmt,2,first)
    dbGetData(hstmt,3,last)
    dbGetData(hstmt,4,street)
    dbGetData(hstmt,5,city)
    dbGetData(hstmt,6,state)
    dbGetData(hstmt,7,zip)
    PRINT "ID:",Address_ID
    PRINT first," ",last
    PRINT street
    IF dbIsNull(hstmt,5) = FALSE
      PRINT city," ",state," ",zip
    ENDIF
    PRINT
  ENDWHILE
ENDIF
```

16.8.20 dbGetDate

Syntax

```
INT = dbGetDate(hstmt as INT,column as INT,date as DBDATE)
```

Description

Gets a DBDATE type directly from a column after a dbGet, dbGetNext or dbGetPrev operation.

Parameters

hstmt - Statement handle returned by dbExecSQL or dbPrepareSQL.

column - Ones based column number to retrieve from.

date - A variable of type DBDATE

Return value

TRUE if data was successfully retrieved and stored in the variable. FALSE on error or empty result set.

Remarks

dbGetData is an alternative to binding a DBDATE variable and is not supported by all database drivers. Certain drivers will not allow directly retrieving data from a column if it is already bound to a variable. The Access driver supports both binding and direct retrieval. Binding variables will always result in faster retrieval of data.

See Also: [dbGetTime](#), [dbGetTimeStamp](#), [dbGetData](#)

Example usage

```
DEF dtAdded as DBDATE
DEF first,last as STRING
DEF Address_ID as INT
...
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses")
IF hstmt
    'bind the columns in the table to appropriate variables
    'column numbers start at 1 starting from the left.

    WHILE dbGet(hstmt)
        dbGetData(hstmt,1,Address_ID)
        dbGetData(hstmt,2,first)
        dbGetData(hstmt,3,last)
        dbGetData(hstmt,8,dtAdded)
        PRINT "ID:",Address_ID
        PRINT "Name:",first,last
        PRINT "Date added:",USING("## ## ####",dtAdded.month,dtAdded.day,dtAdded.year)
    ENDWHILE
ENDIF
```

16.8.21 dbGetErrorCode**Syntax**

STRING = dbGetErrorCode(hstmt as INT)

Description

Returns the ODBC error code for a statement, if any.

Parameters

hstmt - Statement handle created with dbExecSQL or dbPrepareSQL

Return value

A string containing the ODBC error code, or an empty string if no error has occurred.

Remarks

The ODBC error code is a five character string. For human readable text of the error code use dbGetErrorText.

See Also: [dbGetErrorText](#)

Example usage

```
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses")
error = dbGetErrorCode(hstmt)
IF LEN(error)
    PRINT
    PRINT "Error Code: ", error
    PRINT "Error Text: ", dbGetErrorText(hstmt)
    PRINT
ENDIF
```

16.8.22 dbGetErrorText

Syntax

STRING = dbGetErrorText(hstmt as INT)

Description

Returns the ODBC error text for a statement, if any.

Parameters

hstmt - Statement handle created with dbExecSQL or dbPrepareSQL

Return value

A string containing the ODBC error text, or an empty string if no error has occurred.

Remarks

For the ODBC error code use dbGetErrorCode.

See Also: [dbGetErrorCode](#)

Example usage

```
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses")
error = dbGetErrorCode(hstmt)
IF LEN(error)
    PRINT
    PRINT "Error Code: ", error
    PRINT "Error Text: ", dbGetErrorText(hstmt)
    PRINT
ENDIF
```

16.8.23 dbGetFirst

Syntax

INT = dbGetFirst(hstmt as INT)

Description

Gets the first record from a result set.

Parameters

hstmt - Statement handle returned by dbExecSQL or dbPrepareSQL.

Return value

TRUE if the record was retrieved and any bound variables filled in with column data. FALSE if there were no records in the result set.

Remarks

A column that is nullable and currently set to NULL can be determined by `dbIsNull` after a `dbGetFirst` operation. A NULL value can also be determined by checking the optional 'pReturn' variable used when binding the column with `dbBindVariable`. `dbGetFirst` and `dbGetLast` may not work with all database drivers, some text based drivers only support sequential access with the `dbGet`.

See Also: [dbGetNext](#), [dbGetPrev](#), [dbExecSQL](#), [dbPrepareSQL](#), [dbGet](#), [dbGetLast](#)

Example usage

```
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses")
error = dbGetErrorCode(hstmt)
IF LEN(error)
    PRINT
    PRINT "Error Code: ", error
    PRINT "Error Text: ", dbGetErrorText(hstmt)
    PRINT
ENDIF

IF hstmt
    'bind the columns in the table to appropriate variables
    'column numbers start at 1 starting from the left.
    dbBindVariable(hstmt,1,Address_ID)
    dbBindVariable(hstmt,2,first)
    dbBindVariable(hstmt,3,last)
    dbBindVariable(hstmt,4,street)
    dbBindVariable(hstmt,5,city)
    dbBindVariable(hstmt,6,state)
    dbBindVariable(hstmt,7,zip)
    'dbGet returns TRUE if data has been retrieved and stored
    'in the bound variables. Or FALSE if the end of data has been reached or an error
    'skip the first record
    dbGetFirst(hstmt)
    WHILE dbGetNext(hstmt)
        PRINT "ID:",Address_ID
        PRINT first," ",last
        PRINT street
        IF dbIsNull(hstmt,5) = FALSE
            PRINT city, " ",state, " ", zip
        ENDIF
        PRINT
    ENDWHILE
    dbFreeSQL(hstmt)
ENDIF
```

16.8.24 dbGetLast

Syntax

`INT = dbGetLast(hstmt as INT)`

Description

Gets the last record from a result set.

Parameters

hstmt - Statement handle returned by `dbExecSQL` or `dbPrepareSQL`.

Return value

TRUE if the record was retrieved and any bound variables filled in with column data. FALSE if there were no records in the result set.

Remarks

A column that is nullable and currently set to NULL can be determined by `dbIsNull` after a `dbGet` operation. A NULL value can also be determined by checking the optional 'pReturn' variable used when binding the column with `dbBindVariable`. `dbGetFirst` and `dbGetLast` may not work with all database drivers, some text based drivers only support sequential access with the `dbGet`.

See Also: [dbGetNext](#), [dbGetPrev](#), [dbExecSQL](#), [dbPrepareSQL](#), [dbGet](#), [dbGetFirst](#)

Example usage

```
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses")
error = dbGetErrorCode(hstmt)
IF LEN(error)
    PRINT
    PRINT "Error Code: ", error
    PRINT "Error Text: ", dbGetErrorText(hstmt)
    PRINT
ENDIF

IF hstmt
    'bind the columns in the table to appropriate variables
    'column numbers start at 1 starting from the left.
    dbBindVariable(hstmt,1,Address_ID)
    dbBindVariable(hstmt,2,first)
    dbBindVariable(hstmt,3,last)
    dbBindVariable(hstmt,4,street)
    dbBindVariable(hstmt,5,city)
    dbBindVariable(hstmt,6,state)
    dbBindVariable(hstmt,7,zip)
    'dbGet returns TRUE if data has been retrieved and stored
    'in the bound variables. Or FALSE if the end of data has been reached or an error
    'retrieve the last record
    IF dbGetLast(hstmt)
        PRINT "ID:",Address_ID
        PRINT first," ",last
        PRINT street
        IF dbIsNull(hstmt,5) = FALSE
            PRINT city, " ",state, " ", zip
        ENDIF
    PRINT
ENDIF
dbFreeSQL(hstmt)
ENDIF
```

16.8.25 dbGetNext

Syntax

```
INT = dbGetNext(hstmt as INT)
```

Description

Gets the next record from a result set.

Parameters

hstmt - Statement handle returned by `dbExecSQL` or `dbPrepareSQL`.

Return value

TRUE if the record was retrieved and any bound variables filled in with column data. FALSE if the end of the result set was reached or there were no records in the result set.

Remarks

`dbGetNext` and `dbGetPrev` may not work with all database drivers, some text based drivers only support sequential access with the `dbGet` command. In all other cases there is no difference from using `dbGet` or `dbGetNext`. A column that is nullable and currently set to NULL can be determined by `dbIsNull` after a `dbGet` operation. A NULL value can also be determined by checking the optional 'pReturn' variable used when binding the column with `dbBindVariable`.

See Also: [dbGetPrev](#), [dbExecSQL](#), [dbPrepareSQL](#), [dbGet](#)

Example usage

```
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses")
error = dbGetErrorCode(hstmt)
IF LEN(error)
    PRINT
    PRINT "Error Code: ", error
    PRINT "Error Text: ", dbGetErrorText(hstmt)
    PRINT
ENDIF

IF hstmt
    'bind the columns in the table to appropriate variables
    'column numbers start at 1 starting from the left.
    dbBindVariable(hstmt,1,Address_ID)
    dbBindVariable(hstmt,2,first)
    dbBindVariable(hstmt,3,last)
    dbBindVariable(hstmt,4,street)
    dbBindVariable(hstmt,5,city)
    dbBindVariable(hstmt,6,state)
    dbBindVariable(hstmt,7,zip)
    'dbGetNext returns TRUE if data has been retrieved and stored
    'in the bound variables. Or FALSE if the end of data has been reached or an error
    WHILE dbGetNext(hstmt)
        PRINT "ID:",Address_ID
        PRINT first," ",last
        PRINT street
        IF dbIsNull(hstmt,5) = FALSE
            PRINT city," " ,state, " ", zip
        ENDIF
    ENDIF
ENDIF
```

```
        PRINT
    ENDWHILE
    dbFreeSQL (hstmt)
ENDIF
```

16.8.26 dbGetNumCols

Syntax

INT = dbGetNumCols(hstmt as INT)

Description

Returns the number of columns in a result set.

Parameters

hstmt - Statement handle returned by dbExecSQL or dbPrepareSQL

Return value

Number of columns

Remarks

Useful for dynamically binding columns to variables when the number of columns returned is unknown until runtime.

Example usage

```
hstmt = dbExecSQL(pdb, "SELECT AddressID, FirstName, LastName, Address FROM Addresses")
IF hstmt
    pColumns = dbListColumns(pdb, "", hstmt)
    IF pColumns <> NULL
        FOR temp2 = EACH pColumns as STRING
            PRINT #temp2, ", ",
        NEXT
        PRINT
        ListRemoveAll(pColumns, TRUE)
        numcols = dbGetNumCols(hstmt)
        pBindings = NEW(STRING, numcols)
        FOR x = 1 to numcols
            dbBindVariable(hstmt, x, #<STRING>pBindings[x-1, 0])
        NEXT x
        WHILE dbGet(hstmt)
            FOR x=1 to numcols
                PRINT #<STRING>pBindings[x-1, 0], ", ",
            NEXT x
            PRINT
        ENDWHILE
        dbFreeSQL(hstmt)
        DELETE pBindings
    ELSE
        PRINT "No data"
    ENDIF
ENDIF
```

16.8.27 dbGetPrev

Syntax

```
INT = dbGetPrev(hstmt as INT)
```

Description

Gets the previous record from a result set.

Parameters

hstmt - A statement handle returned by `dbExecSQL` or `dbPrepareSQL`

Return value

TRUE if the record was retrieved and any bound variables filled in with column data. FALSE if the beginning of the result set was reached or there were no records in the result set.

Remarks

`dbGetNext` and `dbGetPrev` may not work with all database drivers, some text based drivers only support sequential access with the `dbGet` command. A column that is nullable and currently set to NULL can be determined by `dbIsNull` after a `dbGet` operation. A NULL value can also be determined by checking the optional 'pReturn' variable used when binding the column with `dbBindVariable`.

See Also: [dbGetNext](#), [dbExecSQL](#), [dbPrepareSQL](#), [dbGet](#)

Example usage

```
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses")
error = dbGetErrorCode(hstmt)
IF LEN(error)
    PRINT
    PRINT "Error Code: ", error
    PRINT "Error Text: ", dbGetErrorText(hstmt)
    PRINT
ENDIF

IF hstmt
    'bind the columns in the table to appropriate variables
    'column numbers start at 1 starting from the left.
    dbBindVariable(hstmt,1,Address_ID)
    dbBindVariable(hstmt,2,first)
    dbBindVariable(hstmt,3,last)
    dbBindVariable(hstmt,4,street)
    dbBindVariable(hstmt,5,city)
    dbBindVariable(hstmt,6,state)
    dbBindVariable(hstmt,7,zip)
    'dbGetNext returns TRUE if data has been retrieved and stored
    'in the bound variables. Or FALSE if the end of data has been reached or an error
    WHILE dbGetNext(hstmt)
        PRINT "ID:",Address_ID
        PRINT first," ",last
        PRINT street
        IF dbIsNull(hstmt,5) = FALSE
            PRINT city," " ,state, " ", zip
        ENDIF
    ENDIF
ENDIF
```

```
        PRINT
    ENDWHILE
    'Now print them in reverse order
    WHILE dbGetPrev(hstmt)
        PRINT "ID:",Address_ID
        PRINT first," ",last
        PRINT street
        IF dbIsNull(hStmt,5) = FALSE
            PRINT city, " ", state, " ", zip
        ENDIF
        PRINT
    ENDWHILE
    dbFreeSQL(hstmt)
ENDIF
```

16.8.28 dbGetTime

Syntax

dbGetTime(hstmt as INT,column as INT,time as DBTIME)

Description

Gets a DBTIME type directly from a column after a dbGet, dbGetNext or dbGetPrev operation.

Parameters

hstmt - Statement handle returned by dbExecSQL or dbPrepareSQL.

column - Ones based column number to retrieve from.

time - A variable of type DBTIME

Return value

TRUE if data was successfully retrieved and stored in the variable. FALSE on error or empty result set.

Remarks

dbGetTime is an alternative to binding a DBTIME variable and is not supported by all database drivers. Certain drivers will not allow directly retrieving data from a column if it is already bound to a variable. The Access driver supports both binding and direct retrieval. Binding variables will always result in faster retrieval of data.

See Also: [dbGetDate](#), [dbGetTimeStamp](#), [dbGetData](#)

Example usage

```
DEF tmAdded as DBTIME
DEF first,last as STRING
DEF Address_ID as INT
...
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses")
IF hStmt
    'bind the columns in the table to appropriate variables
    'column numbers start at 1 starting from the left.

    WHILE dbGet(hstmt)
        dbGetData(hstmt,1,Address_ID)
```

```

    dbGetData(hstmt,2,first)
    dbGetData(hstmt,3,last)
    dbGetTime(hstmt,8,tmAdded)
    PRINT "ID:",Address_ID
    PRINT "Name:",first,last
    PRINT "Time added:",USING("## ## ##",tmAdded.hour,tmAdded.minute,tmAdded.second)
ENDWHILE
ENDIF

```

16.8.29 dbGetTimeStamp

Syntax

dbGetTimeStamp(hstmt as INT,column as INT,timestamp as DBTIMESTAMP)

Description

Gets a DBTIMESTAMP type directly from a column after a dbGet, dbGetNext or dbGetPrev operation.

Parameters

hstmt - Statement handle returned by dbExecSQL or dbPrepareSQL.

column - Ones based column number to retrieve from.

timestamp - A variable of type DBTIMESTAMP

Return value

TRUE if data was successfully retrieved and stored in the variable. FALSE on error or empty result set.

Remarks

dbGetTimeStamp is an alternative to binding a DBTIMESTAMP variable and is not supported by all database drivers. Certain drivers will not allow directly retrieving data from a column if it is already bound to a variable. The Access driver supports both binding and direct retrieval. Binding variables will always result in faster retrieval of data.

See Also: [dbGetTime](#), [dbGetDate](#), [dbGetData](#)

Example usage

```

DEF dtAdded as DBTIMESTAMP
DEF first,last as STRING
DEF Address_ID as INT
...
hstmt = dbExecSQL(pdb,"SELECT * FROM Addresses")
IF hstmt
    'bind the columns in the table to appropriate variables
    'column numbers start at 1 starting from the left.

    WHILE dbGet(hstmt)
        dbGetData(hstmt,1,Address_ID)
        dbGetData(hstmt,2,first)
        dbGetData(hstmt,3,last)
        dbGetDate(hstmt,8,dtAdded)
        PRINT "ID:",Address_ID
        PRINT "Name:",first,last
    
```

```
PRINT "Date added:",USING("## ## ####",dtAdded.month,dtAdded.day,dtAdded.year)
PRINT "Time added:",USING("## ## ##",dtAdded.hour,dtAdded.minute,dtAdded.secon
ENDWHILE
ENDIF
```

16.8.30 dbIsNull

Syntax

INT = dbIsNull(hstmt as INT,column as INT)

Description

Tests a column in a record from a result set for NULL.

Parameters

hstmt - A statement handle returned by dbExecSQL or dbPrepareSQL.

column - Ones based column number to test.

Return value

TRUE if the column is nullable and currently set to NULL. FALSE otherwise.

Remarks

An alternative to using the pReturn variable set when binding a column. Some database drivers may not support the dbIsNull command. At least on dbGet or dbGetNext call must have been performed on the result set prior to using this command.

See Also: [dbBindVariable](#)

Example usage

```
WHILE dbGet(hstmt)
PRINT "ID:",Address_ID
PRINT first," ",last
PRINT street
IF dbIsNull(hStmt,5) = FALSE
PRINT city," ",state," ",zip
ENDIF
PRINT
ENDWHILE
```

16.8.31 dbListColumns

Syntax

POINTER = dbListColumns(pConnection as POINTER,tableName as STRING,OPT hStmt as INT)

Description

Returns a linked list containing the names of all of the columns in a table or result set.

Parameters

pConnection - Database pointer returned by dbConnect or dbConnectDSN.

tableName - Name of the table to retrieve the column names

hstmt - Optional. A statement handle returned by `dbExecSQL` or `dbPrepareSQL`.

Return value

A pointer to a linked list of column names or NULL if the table or result set is empty.

Remarks

The pointer returned is a standard IWBASIC linked list type. It must be deleted with `ListRemoveAll` when you are finished enumerating the column names. Each element of the list is a pointer to a string containing a column name.

This command has two forums. For listing all of the column names in a particular table '*hstmt*' must be omitted, or NULL. For listing all of the column names in a result set '*tableName*' must be an empty string "" and '*hstmt*' must be a valid statement handle. In either case the database pointer must be supplied.

Example usage

```
pColumns = dbListColumns(pdb, "", hstmt)
IF pColumns <> NULL
    FOR temp2 = EACH pColumns as STRING
        PRINT #temp2, ", ",
    NEXT
    PRINT
    ListRemoveAll(pColumns, TRUE)
ENDIF

pColumns = dbListColumns(pdb, "USERS")
IF pColumns <> NULL
    FOR temp2 = EACH pColumns as STRING
        PRINT #temp2, ", ",
    NEXT
    PRINT
    ListRemoveAll(pColumns, TRUE)
ENDIF
```

16.8.32 dbListTables

Syntax

POINTER = `dbListTables`(pConnection as POINTER)

Description

Returns a linked list containing the names of all of the tables in a database.

Parameters

pConneciton - A database pointer returned by `dbConnect` or `dbConnectDSN`

Return value

A pointer to a linked list of table names or NULL if the database is empty.

Remarks

The pointer returned is a standard IW BASIC linked list type. It must be deleted with `ListRemoveAll` when you are finished enumerating the table names. Each element of the list is a pointer to a string containing a table name.

Example usage

```
REM List all tables, and their columns
pTables = dbListTables(pdb)
PRINT "Tables:"
IF pTables <> NULL
  FOR temp = EACH pTables as STRING
    PRINT #temp
    PRINT "\tColumns:"
    pColumns = dbListColumns(pdb,#temp)
    IF pColumns <> NULL
      FOR temp2 = EACH pColumns as STRING
        PRINT "\t\t",#temp2
      NEXT
      ListRemoveAll(pColumns,TRUE)
    ENDIF
  NEXT
  ListRemoveAll(pTables,TRUE)
ELSE
  PRINT "Unable to list tables"
ENDIF
```

16.8.33 dbPrepareSQL

Syntax

`INT = dbPrepareSQL(pConnection as POINTER,statement as STRING)`

Description

Prepares an SQL statement for later execution with `dbExecute`.

Parameters

pConnection - A pointer to a database returned by `dbConnect` or `dbConnectDSN`.

statement - The SQL statement to prepare.

Return value

A handle to the prepared SQL statement or `NULL` if a statement handle could not be allocated.

Remarks

A prepared SQL statement is one that has been validated by the driver and converted to the necessary internal representation ready for execution.

Prepared statements are used to perform many duplicate updates or insertions into a database table. Using a prepared statement allows simply updating the bound parameters and calling `dbExecute` for each iteration. The main advantage over direct execution with `dbExecSQL` is speed of updates and insertions.

See Also: [dbExecute](#), [dbFreeSQL](#)

Example usage

```

hstmt = dbPrepareSQL(pdb,"INSERT INTO Addresses (FirstName,LastName,Address) VALUES (?,
IF hstmt
    dbBindParameter(hstmt,1,first,255)
    dbBindParameter(hstmt,2,last,255)
    dbBindParameter(hstmt,3,street,255)
    'after the variables are bound you can insert as many records as needed with one s
    first = "Lisa"
    last = "Jones"
    street = "123 Niagara"
    dbExecute(hstmt)
    '
    first = "Tammy"
    last = "Miller"
    street = "123 America St"
    dbExecute(hstmt)
    '
dbFreeSQL(hstmt)
ENDIF

```

16.9 Appendix**16.9.1 Minimum SQL Grammer****Appendix A - SQL Minimum Grammar**

This section describes the minimum SQL syntax that an ODBC driver must support. The syntax described in this section is a subset of the Entry level syntax of SQL-92.

An application can use any of the syntax in this section and be assured that any ODBC-compliant driver will support that syntax. To determine whether additional features of SQL-92 not in this section are supported, the application should call **SQLGetInfo** with the **SQL_SQL_CONFORMANCE** information type. Even if the driver does not conform to any SQL-92 conformance level, an application can still use the syntax described in this section. If a driver conforms to an SQL-92 level, on the other hand, it supports all syntax included in that level. This includes the syntax in this section because the minimum grammar described here is a pure subset of the lowest SQL-92 conformance level. Once the application knows the SQL-92 level supported, it can determine whether a higher-level feature is supported (if any) by calling **SQLGetInfo** with the individual information type corresponding to that feature.

Drivers that work only with read-only data sources might not support those parts of the grammar included in this section that deal with changing data. An application can determine if a data source is read-only by calling **SQLGetInfo** with the **SQL_DATA_SOURCE_READ_ONLY** information type.

Statement

create-table-statement ::=

```
CREATE TABLE base-table-name  
(column-identifier data-type [, column-identifier data-type]...)
```

Important As a *data-type* in a *create-table-statement*, applications must use a data type from the TYPE_NAME column of the result set returned by **SQLGetTypeInfo**.

```
delete-statement-searched ::=  
DELETE FROM table-name [WHERE search-condition]
```

```
drop-table-statement ::=  
DROP TABLE base-table-name
```

```
insert-statement ::=  
INSERT INTO table-name [( column-identifier [, column-identifier]...)]  
VALUES (insert-value [, insert-value]... )
```

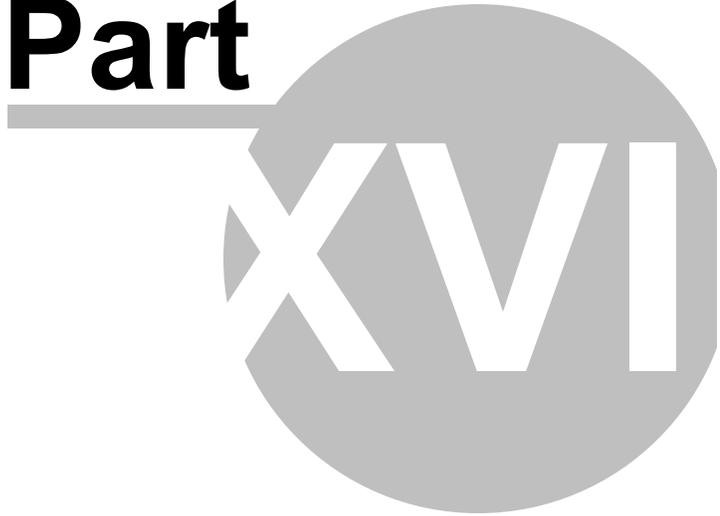
```
select-statement ::=  
SELECT [ALL | DISTINCT] select-list  
FROM table-reference-list  
[WHERE search-condition]  
[order-by-clause]
```

```
statement ::= create-table-statement  
| delete-statement-searched  
| drop-table-statement  
| insert-statement  
| select-statement  
| update-statement-searched
```

```
update-statement-searched  
UPDATE table-name  
SET column-identifier = {expression | NULL }  
[, column-identifier = {expression | NULL}]...  
[WHERE search-condition]
```


Disclaimer / License / Copyright

Part



17 Disclaimer / License / Copyright

License:

By installing/using this product, you are agreeing to be bound by the terms of this document. This Agreement constitutes the complete agreement between you and Ionic Wind Software, the AUTHOR of this software program. If you do not agree to the terms of this Agreement, do not complete this installation.

1. License Grant: In consideration of payment of the license fee, which is part of the price you paid for this product, Ionic Wind Software, AUTHOR, grants to you, USER, a non-exclusive right to use this copy of the software program, *IWBASIC*, SOFTWARE. You may use this product on a single computer. For convenience, the USER may install the licensed copy of the products on other computers owned by the USER for USER's exclusive use or that of immediate family members who live in the same household as the USER, provided that no two of those computers shall ever operate the software at the same time.

The AUTHOR reserves all rights not expressly granted to the USER.

2. Ownership: As the USER, you own the magnetic or other physical media on which the SOFTWARE is originally or subsequently recorded or fixed, but the AUTHOR retains the title to and ownership in the SOFTWARE recorded on the original disk copy and all subsequent copies of the program, regardless of the form or media in or on which the original and other copies may exist. This license is not a sale of the original SOFTWARE or any portion or copy of it.

3. Copy Restrictions: The SOFTWARE and the accompanying materials are copyrighted and contain proprietary information and trade secrets. Unauthorized copying of the SOFTWARE or of the written materials is expressly forbidden. You may be held legally responsible for any infringement of the AUTHOR's intellectual property rights that is caused or encouraged by your failure to abide by the terms of this Agreement. In addition to licensed installations as described above, you may make one (1) copy of the SOFTWARE installation files solely for backup purposes, provided that the copyright notice is reproduced in its entirety on the backup copy.

4. Permitted Uses: This SOFTWARE is licensed to you, the USER, and may not be transferred to any third party for any length of time without the prior written consent of the AUTHOR. You may not modify, adapt, translate, reverse engineer, decompile, disassemble, or sale the SOFTWARE. Written materials provided to you may not be modified, adapted, translated, or used to create derivative works without the prior written consent of the AUTHOR. The USER is granted a royalty-free license to use the SOFTWARE to create their own application and distribute that application freely. However, no portion of any of the component files of the SOFTWARE may be distributed with the USER's application.

5. Termination: This Agreement is effective until terminated. This Agreement will terminate

without notice from the AUTHOR if you fail to comply with any provision contained herein. Upon termination, you shall destroy the written materials, and all original and backup copies of the SOFTWARE, in part, and in whole.

6. Registration: The AUTHOR may from time to time update the SOFTWARE. Updates can be made available to you only if your license has been registered with the AUTHOR.

7. Miscellaneous: This Agreement is governed by the laws of Florida.

Disclaimer and Limited Warranty:

This warranty gives you specific legal rights. You may have other rights, which vary from state to state.

THESE SOFTWARE PRODUCTS, REFERENCE MATERIAL, AND OTHER DOCUMENTATION ARE PROVIDED AS IS. IONIC WIND SOFTWARE, AS LICENSOR, MAKES NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THE USE OF THIS LICENSED MATERIAL, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES ARISING FROM USAGE OF TRADE OR COURSE OF DEALING.

IONIC WIND SOFTWARE SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, OR INCIDENTAL DAMAGES (INCLUDING DAMAGES FROM LOSS OF BUSINESS INFORMATION, AND THE LIKE) ARISING OUT OF THE USE OF OR INABILITY TO USE THESE PRODUCTS, EVEN IF IONIC WIND SOFTWARE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

Copyright:

© 2010 - 2016 Ionic Wind Software All Rights Reserved.

History

Part



18 History

2016-Aug-16	v3.05	<p>Maintenance Release</p> <ul style="list-style-type: none"> fixed - IDE splitter control lines update erratically.
2016-Aug-15	v3.04	<p>Maintenance Release</p> <ul style="list-style-type: none"> fixed - ProjectList Window Caption displayed with no project loaded fixed - Create button in MakeSingle dialog did not have focus when F8 was pressed fixed - Was possible to open second copy of same file when responding to compile error. fixed - File Save As function would allow creation of file with same name with upper case letters. fixed - About dialog displayed incorrect version number for compiler fixed - F1 Help searches were causing the IDE to crash fixed - Multiple Help files were not being sequentially searched correctly fixed - INT, UNIT, INT64, UINT64 did not have numeric ranges describes in variable table Corrected errors in Help file <ul style="list-style-type: none"> Added missing \$IF directive to INDEX Added missing \$IFDEF directive to INDEX Added missing \$OPTION directive to INDEX Added missing \$OPTION directive to INDEX Added missing missing \$THREAD directive to INDEX Added missing \$UNDEF directive to INDEX Added missing 'ENDREGION directive to INDEX Added missing 'REGION directive to INDEX Corrected example of writing FLOAT variable to file Typo in SYSTEM command example Corrected ONEXIT command explanation and example Corrected CHAR/SCHAR entries/examples fixed - Double-clicking REM was not taking USER to proper location in Help file. Added additional info for __IWVER__ constant fixed - Installer was adding unwanted desktop Icon
2015-Aug-6	v3.03	<p>Maintenance Release</p> <ul style="list-style-type: none"> fixed - Single File App was erroneously allowed to have exe placed in different folder than source file fixed - When IDE is first opened the RUN menu option is enabled

		<p>even though no application is loaded.</p> <ul style="list-style-type: none"> fixed - Menu options associated with non-existent debugger left enabled during testing Updated NASM assembler to latest version
2015-Jul-7	v3.02	<p>Maintenance Release</p> <ul style="list-style-type: none"> fixed - Resources were not being linked when <i>Compile/ReLink/Run</i> build option is used with Projects
2015-Jun-27	v3.01	<p>Maintenance Release</p> <ul style="list-style-type: none"> fixed - DATE\$ command declared improperly fixed - typo in iwboptsedit.exe filename during installation
2015-Jun-10	v3.00	<p>Official Release</p>
2015-May-16	v2.516b	<p>Maintenance Release</p> <ul style="list-style-type: none"> fixed - File Extension Association Utility was not working properly fixed - Icon's were not being properly assigned to iwp,iwb,eba, and ewp file types when above utility was used fixed - VersionInfo resource file was not being created properly fixed - Manifest file resource resource was not being read properly
2014-Jan-14	v2.515b	<p>Maintenance Release</p> <ul style="list-style-type: none"> fixed - <i>Find-in-File</i> searches were aborted for files containing a character above ASCII 127 fixed - First control added during a <i>Form Editor</i> session does not display coordinates fixed - Autoindent does not work correctly for the DEFAULT keyword fixed - Can not edit <i>Find-in-File</i> search term directly fixed - The <i>Pick Folder</i> dialog is not modal fixed - <i>Find and Replace</i> is forcing upper case <p>Feature Addition</p> <ul style="list-style-type: none"> Added search path, file type filer and case sensitive status to <i>Output</i> window at the beginning of each <i>Find-in-File</i> search.
2013-Oct-04	v2.514b	<p>Maintenance Release</p> <ul style="list-style-type: none"> fixed - Documentation for <i>Snap to Grid</i> option in <i>Form Editor</i> is missing. <p>Feature Addition</p> <ul style="list-style-type: none"> Added <i>Alignment Aid</i> option to to facilitate alignment of controls with each other.
2013-Aug-01	v2.513b	<p>Maintenance Release</p>

		<ul style="list-style-type: none"> fixed - if the Project List window is closed on IDE startup, some menu options will cause the IDE to lockup requiring Task Manager to close.
2013-Jul-28	v2.512b	<p>Maintenance Release</p> <ul style="list-style-type: none"> fixed - ENTER key wasn't starting FIF search when pressed in searchterm field fixed - compiler options dialog was writing trash into ini file added option to not delete exe file when compile/link fails. Made default to delete. Option temporarily located in IDE Options dialog (will be moved later to Compiler Opts dialog)
2013-Jul-12	v2.511b	<p>Maintenance Release</p> <ul style="list-style-type: none"> fixed - modifications in previous release made edit controls in FIF Output window appear as if they were disabled.
2013-Jul-11	v2.510b	<p>Maintenance Release</p> <ul style="list-style-type: none"> fixed - workspace tab control wasn't resizing correctly when more tabs were present than could be shown fixed - could not compile a static library file fixed - when compiling dll/lib with <i>Compile and Run</i> button the IDE would attempt to execute the file. fixed - with project open but no file open selecting <i>Build\Compile</i> would crash the IDE fixed - Output window operation is erratic when minimizing/maximizing/restoring the IDE. removed ability to position <i>Project List</i> window to right side of IDE added splitter bars to control size of <i>Project List</i> and <i>Output</i> windows
2013-Jun-03	v2.509b	<p>Maintenance Release</p> <ul style="list-style-type: none"> fixed - project list was not updated when project automatically opened on startup fixed - exe was not being deleted when project compile failed fixed - modified include files were being ignored when starting project compile fixed - double-clicking result line in FIF would open 2nd copy of same file. fixed - double-clicking file in project list would open 2nd copy of same file. fixed - could not compile individual source file of project fixed - double-clicking result line in FIF would go to wrong location if there was collapsed text at a lower line number. fixed - double-clicking subroutine in project list would go to

		<p>wrong location if there was collapsed text at a lower line number.</p> <ul style="list-style-type: none"> • fixed - selecting subroutine in Code Editor window combobox would go to wrong location if there was collapsed text at a lower line number. • fixed - "link all" doesn't work right at all • fixed - moved open file tab control from top to bottom portion of workspace to reduce cluttered appearance.
2013-Jun-03	v2.508b	<p>Feature Addition</p> <ul style="list-style-type: none"> • The search term used with F2 (Find Prev), <SHFT>F2 (Find First), F3 (Find Next), <SHFT>F3 (Find Last) can now be established by highlighting the term in the Code window.
2013-Jun-01	v2.507b	<p>Maintenance Release</p> <ul style="list-style-type: none"> • fixed - Tools Menu Editor menu changes bug • fixed - Help Menu Editor menu changes bug • fixed - wasn't picking up assembler options • fixed - eliminated inconsistencies associated with identification of 3 button types. • added following constants: SETID "BN_CLICKED", 0 SETID "BN_DBLCLK", 0x5 SETID "LEFTTEXT", 0x20 SETID "DISABLE", 0x8000000 SETID "SS_SIMPLE", 0xB SETID "SS_LEFT", 0 SETID "SS_CENTER", 0x1 SETID "SS_RIGHT", 0x2 SETID "SS_NOTIFY", 0x100
2013-May-03	v2.506b	<p>Maintenance Release</p> <ul style="list-style-type: none"> • fixed - added 5 missing style flags for buttons • fixed - comments were being capped same as keywords • fixed - mouse wheel will cause print preview to crash • fixed - Tools Menu Editor menu changes do not appear until restart • fixed - Help Menu Editor menu changes do not appear until restart • fixed - Open files do not appear in Windows menu
2013-Mar-06	v2.503b	<p>Beta Release</p> <ul style="list-style-type: none"> • New IDE /w enhanced features • Major Revision to Help File (WIP) • Corrected documentation for \$REGION, \$SENDREGION,

		'REGION, 'ENDREGION
2011-Sep-16	v2.09	Maintenance Release <ul style="list-style-type: none">• Numerous bug fixes• Conversion to standard release format• Minor edits to Help file.
2011-Mar-01	v2.00	Initial Release

