

IBM VisualAge for C++ for Windows

S33H-5031-00

*User's Guide*

Version 3.5





IBM VisualAge for C++ for Windows

S33H-5031-00

*User's Guide*

Version 3.5

**Note!**

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xxix.

## **First Edition (February 1996)**

This edition applies to Version 3.5 of IBM VisualAge for C++ for Windows (33H4979, 33H4980) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for readers’ comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Canada Ltd. Laboratory  
Information Development  
2G/345/1150/TOR  
1150 Eglinton Avenue East  
North York, Ontario, Canada M3C 1H7

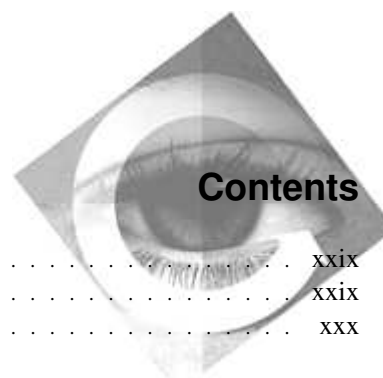
You can also send your comments by facsimile (attention: RCF Coordinator), or you can send your comments electronically to IBM. See “Communicating Your Comments to IBM” for a description of the methods. This page immediately precedes the Readers’ Comment Form at the back of this publication.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1992, 1996. All rights reserved.

© Copyright INTERSOLV, Inc., 1995. INTERSOLV DataDirect ODBC Drivers and related documentation.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.



<b>Notices</b> . . . . .	xxix
Programming Interface Information . . . . .	xxix
Trademarks and Service Marks . . . . .	xxx

<b>About This Guide</b> . . . . .	xxxi
Who Should Read This Guide . . . . .	xxxi
How to Read Syntax Diagrams . . . . .	xxxi
Syntax for Commands, Preprocessor Directives, and Statements . . . . .	xxxi
Syntax for Compiler Options . . . . .	xxxiv
How to Get Help . . . . .	xxxiv
Getting Help Inside VisualAge for C++ . . . . .	xxxv
Getting Help from the Command Line . . . . .	xxxvi
Getting Help for a Keyword or Construct . . . . .	xxxvi
Online Documents Available in VisualAge for C++ . . . . .	xxxvi

---

<b>Part 1. Developing with WorkFrame</b> . . . . .	1
----------------------------------------------------	---

<b>Chapter 1. Introducing WorkFrame</b> . . . . .	3
Overview . . . . .	3
Opening a Project . . . . .	4
Working with Projects . . . . .	4
Getting Help . . . . .	7
Using Contextual Help . . . . .	7
Using <b>How Do I?</b> Information . . . . .	7

<b>Chapter 2. Managing Projects</b> . . . . .	9
Introducing Projects . . . . .	9
Project Parts . . . . .	9
Actions . . . . .	9
Project Views . . . . .	10
Icon View . . . . .	10
Tree View . . . . .	14
Creating a Project . . . . .	15
Project Smarts . . . . .	16
Creating Projects from Project Smarts . . . . .	17
Organizing Projects . . . . .	19
Project Geometry . . . . .	23

<b>Chapter 3. Project and Tool Option Setup</b> . . . . .	25
-----------------------------------------------------------	----

Project Settings . . . . .	25
Project Settings - Target Page . . . . .	26
Project Settings - Directories Page . . . . .	28
Project Settings - Environment page . . . . .	30
Project Settings - Name Page . . . . .	31
Tools Options Setup . . . . .	32
Tools Options . . . . .	32
Build Smarts . . . . .	34
 <b>Chapter 4. Building Your Target . . . . .</b>	 37
Build and Make . . . . .	37
The Build Utility . . . . .	38
Build Prerequisites . . . . .	38
Setting Build Options . . . . .	39
Build Options - Actions Page . . . . .	40
Build Options - Make Page . . . . .	42
Build Options - Project Page . . . . .	44
Build Options - Display Page . . . . .	46
Running Build from the Command Line . . . . .	48
The MakeMake Utility . . . . .	52
Generating Makefiles . . . . .	52
Using MakeMake from the Command Line . . . . .	55
<hr/>	
<b>Part 2. Compiling Your Program . . . . .</b>	<b>57</b>
 <b>Chapter 5. Starting the Compiler . . . . .</b>	 59
Compiling within WorkFrame . . . . .	59
Compiling from the Command Line . . . . .	60
Using Response Files . . . . .	61
Compiling from a Makefile . . . . .	62
 <b>Chapter 6. Controlling Compiler Input . . . . .</b>	 63
Compiling Programs with Multiple Source Files . . . . .	63
File Types . . . . .	64
Using Wildcards in Filenames . . . . .	65
Windows Environment Variables for Compiling . . . . .	66
Filenames in ICC . . . . .	67
Controlling <b>#include</b> Search Paths . . . . .	68
<b>#include</b> Syntax . . . . .	68
<b>#include</b> File Name Syntax . . . . .	68
Ways to Control the <b>#include</b> Search Paths . . . . .	69
<b>#include</b> Search Order . . . . .	70
Accumulation of Options . . . . .	70

Setting the Source Code Language Level . . . . .	71
ANSI . . . . .	72
SAA Level 2 . . . . .	72
Extended . . . . .	73
Compatible . . . . .	73
<b>Chapter 7. Controlling Compiler Output . . . . .</b>	<b>75</b>
Object Files . . . . .	76
Executable Files . . . . .	78
Compiler Listings . . . . .	79
Temporary Files . . . . .	79
Messages . . . . .	80
Return Codes . . . . .	80
Precompiled Header Files . . . . .	81
Using the Intermediate Code Linker . . . . .	81
Intermediate Code Files . . . . .	82
Restrictions . . . . .	83
Using the /Gu Option . . . . .	84
Error Checking . . . . .	84
Inlining User Code . . . . .	85
Using Keywords . . . . .	85
Using the /Oi Option . . . . .	86
Benefits of Inlining . . . . .	88
Drawbacks of Inlining . . . . .	89
Restrictions on Inlining . . . . .	89
Setting the Calling Convention . . . . .	90
Choosing Your Runtime Libraries . . . . .	91
Static and Dynamic Linking . . . . .	92
Using the Multithread Library . . . . .	93
Enabling Subsystem Development . . . . .	94
Using Precompiled Headers . . . . .	94
Determining the Initial Sequence . . . . .	95
Matching the Initial Sequence . . . . .	96
Using Multiple Initial Sequences . . . . .	98
Organizing Your Source Files . . . . .	100
Controlling the Logo Display on Compiler Invocation . . . . .	102
Controlling Stack Allocation and Stack Probes . . . . .	102
Using Stack Probes . . . . .	103
Setting the Stack Size . . . . .	105
<b>Chapter 8. Setting Compiler Options . . . . .</b>	<b>107</b>
Specifying Compiler Options . . . . .	107
Setting Options on the Command Line . . . . .	107

Setting Options in ICC . . . . .	107
Setting Options in the WorkFrame Environment . . . . .	108
Using Parameters with Compiler Options . . . . .	109
Strings . . . . .	109
Filenames . . . . .	110
Switches . . . . .	110
Numbers . . . . .	110
Scope of Compiler Options . . . . .	111
Specifying Options with Multiple Source Files . . . . .	111
ICC Combined with Options Entered on the Command Line . . . . .	112
Related Options . . . . .	112
Conflicting Options . . . . .	112
Language-Dependent Options . . . . .	113
Compiler Options for Windows Programming . . . . .	114
Examples of Compiler Options for Choosing Libraries . . . . .	115
Compiler Option Classification . . . . .	116
Compiler Options Summary . . . . .	117
Output File Management Options . . . . .	121
/Fa . . . . .	122
/Fb . . . . .	123
/Fc . . . . .	123
/Fe . . . . .	124
/Fi . . . . .	124
/FI . . . . .	125
/Fm . . . . .	125
/Fo . . . . .	126
/Ft . . . . .	126
/Fw . . . . .	127
#include File Search Options . . . . .	128
/I . . . . .	128
/Xc . . . . .	129
/Xi . . . . .	129
Listing File Options . . . . .	130
/L . . . . .	131
/La . . . . .	131
/Lb . . . . .	132
/Lc . . . . .	132
/Lf . . . . .	132
/Li . . . . .	133
/Lj . . . . .	133
/Lp . . . . .	133
/Ls . . . . .	134
/Lt . . . . .	134



/Lu	134
/Lx	135
/Ly	135
Debugging and Diagnostic Information Options	135
/N	135
/qdbgunref	136
/Ti	136
/Tm	137
/Tn	137
/Tx	138
/W	138
/Wgrp	138
Source Code Options	143
/J	143
/qbitfields	143
/qdigraph	143
/qlonglong	144
/S	144
/Sd	144
/Sg	145
/Sh	146
/Si	146
/Sm	147
/Sn	147
/Sp	147
/Sq	148
/Sr	148
/Ss	148
/Su	149
/Sv	149
/Tc	149
/Td	150
/Tp	150
Preprocessor Options	151
/D	151
/P	152
/Pc	152
/Pd	152
/Pe	153
/U	153
Code Generation Options	154
/G	154
/Gd	155

/Ge	155
/Gf	156
/Gh	156
/Gi	157
/Gl	157
/Gm	158
/Gn	158
/Gs	158
/Gu	159
/Gw	159
/Gx	160
/M	160
/Nd	161
/Nt	161
/Nx	162
/O	162
/Oc	163
/Oi	163
/Ol	164
/Om	164
/Op	164
/Os	165
/qalias	165
/qisolated_call	165
/qtune	166
/qwin32s	166
/R	167
System Object Model (SOM) Options	168
/Ga	168
/Gb	169
/Gz	169
/Xs	169
/Fr	170
/Fs	170
Other Options	171
/?	171
/B	172
/C	172
/H	172
/Q	173
/qautoimported	173
/qautothread	173
/qignprag	174

/qlibansi . . . . .	174
/qmakedep . . . . .	174
/qro . . . . .	175
/qsomvolattr . . . . .	175
/V . . . . .	175
Options for New ANSI Standards . . . . .	176
/qrtti . . . . .	176

---

## **Part 3. Linking Your Program . . . . . 177**

### **Chapter 9. Starting the Linker . . . . . 179**

Linking within WorkFrame . . . . .	179
Linking from the Command Line . . . . .	180
Using Response Files . . . . .	181
Linking through the Compiler . . . . .	182
Linking from a Make File . . . . .	183

### **Chapter 10. Optimized Linking . . . . . 185**

### **Chapter 11. Input and Output . . . . . 187**

Search Rules . . . . .	187
Specifying Directories . . . . .	188
Filename Defaults . . . . .	189
Specifying Object Files . . . . .	189
Specifying Executable Output Type . . . . .	190
Producing an .exe File . . . . .	190
Producing a Dynamic Link Library . . . . .	191
Generating a Map File . . . . .	191
Linker Return Codes . . . . .	192

### **Chapter 12. Linking with Library Files . . . . . 193**

Linking with .lib Files . . . . .	194
Linking to Dynamic Link Libraries . . . . .	194

### **Chapter 13. Setting Linker Options . . . . . 197**

Setting Options on the Command Line . . . . .	197
Setting Options in the ILINK Environment Variable . . . . .	198
Setting Options in the WorkFrame Environment . . . . .	198
Specifying Numeric Arguments . . . . .	199
Summary of Linker Options . . . . .	201
Linker Options . . . . .	202
/? . . . . .	202
/ALIGNADDR . . . . .	202

/ALIGNFILE . . . . .	203
/BASE . . . . .	203
/BROWSE, NOBROWSE . . . . .	204
/CODE . . . . .	204
/DATA . . . . .	205
/DBGPACK, /NODBGPACK . . . . .	205
/DEBUG, /NODEBUG . . . . .	206
/DEFAULTLIBRARYSEARCH, /NODEFAULTLIBRARYSEARCH . . . .	206
/DLL . . . . .	207
/ENTRY . . . . .	207
/EXECUTABLE . . . . .	207
/EXTDICTIONARY, /NOEXTDICTIONARY . . . . .	208
/FIXED, /NOFIXED . . . . .	208
/FORCE . . . . .	209
/HEAP . . . . .	209
/HELP . . . . .	209
/INCLUDE . . . . .	209
/INFORMATION, /NOINFORMATION . . . . .	210
/LINENUMBERS, /NOLINENUMBERS . . . . .	210
/LOGO, /NOLOGO . . . . .	210
/MAP, /NOMAP . . . . .	211
/OPTFUNC, /NOPTFUNC . . . . .	211
/OUT . . . . .	212
/PMTYPE . . . . .	212
/SECTION . . . . .	213
/SEGMENTS . . . . .	213
/STACK . . . . .	214
/STUB . . . . .	214
/SUBSYSTEM . . . . .	215
/VERBOSE . . . . .	215
/VERSION . . . . .	215
<hr/>	
<b>Part 4. Editing Files . . . . .</b>	<b>217</b>
<b>Chapter 14. Introduction to the Editor . . . . .</b>	<b>219</b>
Creating a New File . . . . .	219
Entering and Editing Text . . . . .	220
Undoing Changes . . . . .	223
Saving a File . . . . .	224
Closing a Document View . . . . .	225
Opening an Existing File . . . . .	225
Finding Text . . . . .	226
Finding and Replacing Text . . . . .	227

Finding a Specific Line in the File . . . . .	228
Creating and Finding Marks . . . . .	229
Creating and Naming a Mark . . . . .	229
Finding a Named Mark . . . . .	230
Using a Quick Mark . . . . .	231
Embedding Another File in the Current Document . . . . .	232
Issuing Editor Commands . . . . .	233
Issuing Multiple Editor Commands . . . . .	234
More on Issuing Editor Commands . . . . .	234
Blocking and Manipulating Text . . . . .	235
Editor Block Manipulation Facilities . . . . .	235
Unmarking a Block of Text . . . . .	236
Marking Blocks of Text . . . . .	236
Manipulating Marked Blocks . . . . .	238
Working with Multiple Document Views . . . . .	239
Using a Parser . . . . .	246
Displaying Different Parts of a Program . . . . .	247
Entering Some Code . . . . .	248
 <b>Chapter 15. Customizing the Editor</b> . . . . .	 249
Using the Editor Tool Bar . . . . .	249
Changing Tool Bar Display Characteristics . . . . .	249
Adding Your Own Items to the Tool Bar . . . . .	250
Adding Your Own Items to the Menu Bar . . . . .	251
Customizing the Keyboard . . . . .	252
Remapping the Keyboard Using the Set Key Command . . . . .	252
Remapping the Keyboard Using the Set Action Command . . . . .	253
Customizing the Autosave Facility for the Editor . . . . .	253
Changing Editor Tab Settings . . . . .	255
Changing the Base Editor Font . . . . .	256
Changing Token Display Attributes . . . . .	257
Modifying Editor Behavior Permanently . . . . .	258
Standard Editor Profiles . . . . .	258
User-Defined Load Profiles . . . . .	259
Storing Personalized Profiles . . . . .	259
Sample Personalized Profile . . . . .	260

---

## Part 5. Debugging Your Program . . . . . 261

<b>Chapter 16. Introduction</b> . . . . .	263
Hardware Requirements . . . . .	263
Software Requirements . . . . .	263
Compiling and Linking Your Program . . . . .	263

<b>Chapter 17. Getting Started</b> . . . . .	265
Debugging Win32 Applications on Windows NT or Windows 95 . . . . .	265
Debugging Win32s Applications . . . . .	266
Understanding the Search Path . . . . .	268
Limitations . . . . .	268
Using the Process List Window . . . . .	268
Starting the Debugger from WorkFrame . . . . .	269
Ending the Debugging Session . . . . .	270
 <b>Chapter 18. Frequently Used Features of the Debugger</b> . . . . .	271
Understanding the Debug on Demand Feature . . . . .	271
Using the Tool Bar . . . . .	271
Executing a Program . . . . .	273
Setting Breakpoints . . . . .	274
 <b>Chapter 19. Introducing the Main Debugging Windows</b> . . . . .	275
Using the Debug Session Control Window . . . . .	275
File Menu Choices . . . . .	277
Breakpoints Menu Choices . . . . .	279
Monitors Menu Choices . . . . .	286
Run Menu Choices . . . . .	287
Options Menu Choices . . . . .	288
Windows Menu Choices . . . . .	293
Help Menu Choices . . . . .	293
Using the Source Windows . . . . .	293
File Menu Choices . . . . .	296
View Menu Choices . . . . .	296
Breakpoints Menu Choices . . . . .	299
Monitors Menu Choices . . . . .	300
Run Menu Choices . . . . .	301
Options Menu Choices . . . . .	303
Windows Menu Choices . . . . .	304
Help Menu Choices . . . . .	304
 <b>Chapter 20. Introducing the Other Debugging Windows</b> . . . . .	305
Using the Call Stack Window . . . . .	305
File Menu Choice . . . . .	306
Options Menu Choices . . . . .	306
Windows Menu Choices . . . . .	307
Help Menu Choices . . . . .	307
Using the Register Window . . . . .	308
File Menu Choices . . . . .	308
Options Menu Choice . . . . .	308

Windows Menu Choices . . . . .	309
Help Menu Choices . . . . .	309
Using the Storage Window . . . . .	309
File Menu Choice . . . . .	310
Options Menu Choices . . . . .	310
Windows Menu Choices . . . . .	313
Help Menu Choices . . . . .	313
Using the Local Variable Window . . . . .	313
File Menu Choice . . . . .	313
Edit Menu Choices . . . . .	314
Options Menu Choices . . . . .	315
Windows Menu Choices . . . . .	316
Help Menu Choices . . . . .	316
Using the Monitor Windows . . . . .	316
Using the Breakpoint List Window . . . . .	316
File Menu Choice . . . . .	317
Edit Menu Choices . . . . .	317
Set Menu Choices . . . . .	318
Options Menu Choices . . . . .	318
Windows Menu Choices . . . . .	320
Help Menu Choices . . . . .	320
 <b>Chapter 21. Expressions Supported . . . . .</b>	 323
Supported Expression Operands . . . . .	323
Supported Expression Operators . . . . .	324
Supported Data Types . . . . .	325
<hr/>	
<b>Part 6. Browsing Programs and Libraries . . . . .</b>	<b>327</b>
 <b>Chapter 22. Overview . . . . .</b>	 329
Understanding the Browser . . . . .	329
Concepts Used by the Browser . . . . .	330
Using the Mouse . . . . .	331
Getting Help While You Are Using the Browser . . . . .	331
Using Contextual Help . . . . .	331
Using the How Do I... Information . . . . .	332
 <b>Chapter 23. Getting Started . . . . .</b>	 333
Starting the Browser . . . . .	333
From the command line . . . . .	333
From the Program Manager . . . . .	334
From the IBM WorkFrame environment . . . . .	334
From the VisualAge for C++ Debugger . . . . .	334

Creating Files to Use with the Browser . . . . .	335
Closing the Browser . . . . .	336
<b>Chapter 24. Understanding and Using the Browser User Interface . . . . .</b>	<b>339</b>
The List Window . . . . .	340
Types of List Windows . . . . .	341
Browsing List Objects . . . . .	343
Understanding Browser Generated Flags . . . . .	344
Changing the Default List Window Settings . . . . .	345
Printing and Saving your Lists . . . . .	350
The Graph Window . . . . .	351
Getting a Graph Overview . . . . .	354
Organizing the Graph . . . . .	354
Selecting a Graph Zone . . . . .	355
Browsing Graph Objects . . . . .	356
Changing the Default Graph Window Settings . . . . .	356
Printing and Saving your Graphs . . . . .	361
Changing Browser Settings . . . . .	363
Changing Paths Used by the Browser . . . . .	363
Changing Help Levels . . . . .	365
Changing Fonts . . . . .	366
Loading Files into the Browser . . . . .	368
Merging Files . . . . .	369
Finding Objects in the Current Window . . . . .	371
Searching for Objects in the Entire Browser Database . . . . .	372
The History Window . . . . .	373
<b>Chapter 25. Using the Browser . . . . .</b>	<b>375</b>
Using the Browser to Assist in Development . . . . .	375
Editing and Viewing Source Files . . . . .	375
Browsing without Recompiling . . . . .	376
Browsing the IBM VisualAge for C++ Open Class Library . . . . .	376
Browsing More Than One Program or Library at a Time . . . . .	377
Showing VisualAge for C++ Open Class Library Documentation . . . . .	377
Using the Browser to Aid Program Understanding . . . . .	378
List All Classes Defined in the Currently Loaded Program . . . . .	378
List All Files Used to Create the Currently Loaded Program . . . . .	378
Listing All Objects Defined in a File . . . . .	379
Listing Implementing Files . . . . .	380
Listing All Friends of a Class . . . . .	380
Listing All Friendships of a Class or Function . . . . .	381
Listing Immediate Callers and Callees for a Function . . . . .	382
Listing All Class Members . . . . .	383



Listing Overriding Derived Classes . . . . .	384
Listing Instantiations of Classes or Functions . . . . .	385
Listing All the Exceptions That A Function May Encounter . . . . .	385
Viewing Class Relationships . . . . .	386
Viewing Call Chains . . . . .	387
Viewing Include File Relationships . . . . .	388
Using QuickBrowse . . . . .	389
What Do You See When QuickBrowse Starts . . . . .	390
Scenarios for Using QuickBrowse . . . . .	391
Updating the Browser Database . . . . .	392
Adding Menu Items to the Load ► and Merge ► Cascade menus . . . . .	393
 <b>Chapter 26. A Tour of the Browser</b> . . . . .	 395
Starting the Browser and Loading User Interface Classes . . . . .	396
Finding A Class . . . . .	397
Showing the Inheritance Relationship of a Class . . . . .	398
Finding Another Class . . . . .	399
Changing the View of a Graph . . . . .	400
Investigating the Members of a Class . . . . .	401
Customizing Program Elements . . . . .	402
Editing Files from the Browser . . . . .	402
Organizing the Information in a List Window . . . . .	404
Finding A Function . . . . .	405
Showing the VisualAge for C++ Documentation for a Particular Function . . . . .	406
More About the PopUp Menu Actions . . . . .	406
Invoking Actions Again . . . . .	407
Graphing Include File Relationships . . . . .	407
Returning to Previous Queries/Displays . . . . .	408
Keeping Your Windows From Being Replaced . . . . .	409
Changing the Default Settings for List and Graph Windows . . . . .	410
Manipulating Graphs . . . . .	410
The Browser and WorkFrame . . . . .	411
 <b>Chapter 27. Troubleshooting</b> . . . . .	 413
The Browser Won't Start . . . . .	413
Error Loading a .EXE, .DLL, or .LIB file . . . . .	413
Error Loading a .PDB File . . . . .	413
Adding Files to the Load ► and Merge ► Menus Doesn't Work . . . . .	413
The Graph Zone Will Not Maximum Zoom . . . . .	414
 <b>Chapter 28. Browser Fast-Path Keys and Menu Descriptions</b> . . . . .	 415
Fast-Path Keys . . . . .	415
PullDown Menus . . . . .	416

File PullDown Menu . . . . .	417
Edit PullDown Menu . . . . .	418
View PullDown Menu . . . . .	419
Actions PullDown Menu . . . . .	420
Options PullDown Menu . . . . .	421
Order PullDown Menu . . . . .	422
Windows PullDown Menu . . . . .	422
Project PullDown Menu . . . . .	423
Help PullDown Menu . . . . .	423
PopUp Menus . . . . .	424
PopUp Menu Items for List and Graph Windows . . . . .	424
Object PopUp Menu Items . . . . .	425

---

## **Part 7. Performance Execution Trace Analyzer . . . . . 429**

### **Chapter 29. Introducing the Performance Execution Trace Analyzer . . . . 431**

### **Chapter 30. Preparing Your Program for the Performance Analyzer . . . . 433**

Compiling and Linking Your Program . . . . .	433
Compiling . . . . .	433
Linking . . . . .	433
Tracing Dynamic Link Libraries (DLLs) . . . . .	434
Tracing System Calls . . . . .	434
Creating User Events in Your Program . . . . .	435
Starting and Stopping the Performance Analyzer from Your Program . . . . .	436
Understanding Overhead Time . . . . .	438

### **Chapter 31. Starting the Performance Analyzer from a Command Line . . 439**

Tracing an Executable . . . . .	439
Analyzing an Existing Trace File . . . . .	440
Displaying the Performance Analyzer's Main Control Window . . . . .	440

### **Chapter 32. Starting the Performance Analyzer from WorkFrame . . . . 441**

### **Chapter 33. Exiting the Performance Analyzer . . . . . 443**

### **Chapter 34. Creating a Trace File . . . . . 445**

### **Chapter 35. Creating a Customized Trace File . . . . . 447**

Enabling and Disabling Components . . . . .	447
Selecting the Call Depth for Each Thread . . . . .	449
Using Time Stamps . . . . .	449
Setting and Removing Triggers . . . . .	449

Enabling and Disabling Buffer Wrap . . . . .	450
Naming the Trace File . . . . .	450
<b>Chapter 36. Saving Trace File Settings . . . . .</b>	<b>453</b>
<b>Chapter 37. Using the Performance Analyzer Diagrams . . . . .</b>	<b>455</b>
Opening a Trace File in a Diagram . . . . .	456
<b>Chapter 38. Introducing the Performance Analyzer Windows . . . . .</b>	<b>457</b>
Performance Analyzer - Specify Profile Location Window . . . . .	457
Performance Analyzer - Window Manager Window . . . . .	458
Areas of the Performance Analyzer - Window Manager Window . . . . .	459
Create Trace Window . . . . .	462
Areas of the Create Trace Window . . . . .	463
Trace Generation Window . . . . .	463
Areas of the Trace Generation Window . . . . .	464
Application Monitor Window . . . . .	470
Areas of the Application Monitor Window . . . . .	470
Analyze Trace Window . . . . .	472
Areas of the Analyze Trace Window . . . . .	473
<b>Chapter 39. Viewing Trace Files . . . . .</b>	<b>475</b>
Using Filtering . . . . .	475
Using Zooming . . . . .	476
Using Scaling . . . . .	476
Using Scrolling . . . . .	476
Using Multiple Views . . . . .	477
Recognizing Patterns . . . . .	477
Understanding Correlation . . . . .	477
<b>Chapter 40. Call Nesting Diagram . . . . .</b>	<b>479</b>
Areas of the Call Nesting Diagram . . . . .	480
Call Nesting Menu Bar Summary . . . . .	480
Call Nesting Status Area . . . . .	483
Call Nesting Pop-up Menus . . . . .	483
<b>Chapter 41. Dynamic Call Graph . . . . .</b>	<b>485</b>
Dynamic Call Graph Nodes and Arcs . . . . .	487
Displaying Information about Nodes and Arcs . . . . .	487
Node and Arc Color . . . . .	488
Node Size . . . . .	489
Areas of the Dynamic Call Graph . . . . .	489
Dynamic Call Graph Menu Bar Summary . . . . .	489

Dynamic Call Graph Status Area . . . . .	492
Dynamic Call Graph Pop-up Menus . . . . .	492
Dynamic Call Graph Zoom Bar . . . . .	496
<b>Chapter 42. Execution Density Diagram . . . . .</b>	<b>499</b>
Areas of the Execution Density Diagram . . . . .	500
Execution Density Menu Bar Summary . . . . .	500
Execution Density Status Area . . . . .	503
Execution Density Pop-up Menus . . . . .	503
Execution Density Current Column Indicator . . . . .	504
Execution Density Vertical Ruler . . . . .	504
<b>Chapter 43. Statistics Diagram . . . . .</b>	<b>505</b>
Areas of the Statistics Diagram . . . . .	506
Statistics Menu Bar Summary . . . . .	506
Statistics Summary Pane . . . . .	509
Statistics Details Pane . . . . .	509
<b>Chapter 44. Time Line Diagram . . . . .</b>	<b>511</b>
Areas of the Time Line Diagram . . . . .	512
Time Line Menu Bar Summary . . . . .	512
Time Line Status Area . . . . .	515
Time Line Pop-up Menus . . . . .	515
Time Line Vertical Ruler . . . . .	516
<hr/>	
<b>Part 8. Managing Libraries . . . . .</b>	<b>517</b>
<b>Chapter 45. Using IBM Library Manager . . . . .</b>	<b>519</b>
Running ILIB . . . . .	519
Using the Command Line . . . . .	520
Using the ILIB Environment Variable . . . . .	520
Using an ILIB Response File . . . . .	521
Specifying ILIB Parameters - Examples . . . . .	522
Controlling ILIB Input . . . . .	523
Controlling ILIB Output . . . . .	523
Controlling ILIB Output - Examples . . . . .	524
ILIB Objects . . . . .	525
Summary of ILIB Objects . . . . .	525
Add/Replace Object . . . . .	526
/EXTRACT . . . . .	527
/REMOVE . . . . .	527
ILIB Options . . . . .	528
Summary of ILIB Options . . . . .	528

/?	529
/BACKUP, /NOBACKUP	529
/BROWSE, /NOBROWSE	530
/DEF	530
/FREEFORMAT, /NOFREEFORMAT	530
/GENDEF	530
/GENIMPLIB	531
/HELP	531
/LIST	531
/NOEXTDICTIONARY, /EXTDICTIONARY	531
/OUT	532
/QUIET, /NOLOGO, /LOGO, /NOQUIET	532
/WARN, /NOWARN	532
<b>Chapter 46. Module Definition Files</b>	533
Reserved Words	534
Summary of Module Statements	536
Module Statements	536
BASE	536
DESCRIPTION	537
EXPORTS	537
HEAPSIZE	539
LIBRARY	540
NAME	540
STACKSIZE	541
STUB	542
VERSION	542
<b>Chapter 47. Packaging the VisualAge for C++ Runtime DLLs</b>	543
Using the DLLRNAME Utility	544
How DLLRNAME Works	545
What DLLRNAME Will Not Do	545
Other Uses for DLLRNAME	545
DLLRNAME Options	546
/H (Help)	546
/N (Do Not Rename DLL)	546
/Q (Do Not Display Logo)	546
/R (Do Not Generate Report)	547
An Example	547
<b>Part 9. Using the Data Access Builder</b>	549
<b>Chapter 48. Introduction</b>	551

Overview . . . . .	551
Supported Database Products . . . . .	552
Setting Up a Data Access Builder WorkFrame Project . . . . .	552
Starting Data Access Builder . . . . .	553
Opening a Previously Saved Data Access Builder Session . . . . .	553
Migrating a Previously Saved Data Access Builder Session . . . . .	554
The Data Access Builder Window . . . . .	554
Startup Window . . . . .	555
Customizing the Display of Mappings . . . . .	556
Icon Size . . . . .	556
Horizontal/Vertical Orientation . . . . .	556
Registering Database Products . . . . .	556
Steps Involved in Creating Classes . . . . .	557
 <b>Chapter 49. Creating Table and Class Objects . . . . .</b>	 559
Accessing Database Tables/Views . . . . .	559
Selecting the Code Generation Options . . . . .	561
Specifying Default Generation Options . . . . .	563
Identifying the Data Access Builder Icons . . . . .	565
Database Table Icons . . . . .	565
Class Object Icons . . . . .	565
Generating Different Kinds of Classes for the Same Table/View . . . . .	566
Deleting Class and Table/View Objects . . . . .	567
Viewing the Table/View Settings . . . . .	568
Table/View Page . . . . .	568
Columns Page . . . . .	569
Foreign Keys Page . . . . .	570
Changes in the Database Table/View Definition . . . . .	571
 <b>Chapter 50. Generating the Source Code . . . . .</b>	 573
Customizing the Class Mapping . . . . .	573
Class Settings Notebook - Names Page . . . . .	573
Class Settings Notebook - Attributes Page . . . . .	574
Class Settings Notebook - Members Page . . . . .	577
Generating the Class Source Code . . . . .	578
Saving a Data Access Builder Session . . . . .	579
Saving a Data Access Builder Session under Another Name . . . . .	579
Viewing Generated Files . . . . .	580
Using the Generated Files . . . . .	580
 <b>Chapter 51. Getting Started with ODBC . . . . .</b>	 581
About ODBC Drivers . . . . .	581
ODBC Supported Database Systems . . . . .	581

Installing the ODBC Drivers . . . . .	582
Driver names . . . . .	582
Configuring Data Sources . . . . .	582
Windows 3.x Run-time Support . . . . .	582
Error Messages . . . . .	582
Locking and Isolation Levels . . . . .	584
Locking . . . . .	584
Isolation Levels . . . . .	584
<b>Chapter 52. Supported ODBC Drivers . . . . .</b>	<b>587</b>
DB2 Driver . . . . .	587
Oracle 7 Driver . . . . .	588
System Requirements . . . . .	588
Configuring Data Sources . . . . .	588
Connecting to a Data Source Using a Logon Dialog Box . . . . .	589
Connecting to a Data Source Using a Connection String . . . . .	589
Oracle 7 Data Types . . . . .	591
Isolation and Lock Levels Supported . . . . .	592
ODBC Implementation Level . . . . .	592
Number of Connections and Statements Supported . . . . .	592
Sybase System 10 Driver . . . . .	593
System Requirements . . . . .	593
Configuring Data Sources . . . . .	593
Connecting to a Data Source Using a Logon Dialog Box . . . . .	596
Connecting to a Data Source Using a Connection String . . . . .	596
Data Types . . . . .	599
Isolation and Lock Levels Supported . . . . .	600
ODBC Conformance Level . . . . .	600
Number of Connections and Statements Supported . . . . .	601
<hr/>	
<b>Part 10. Defining National Characteristics . . . . .</b>	<b>603</b>
<b>Chapter 53. Code Set Conversion Utilities . . . . .</b>	<b>605</b>
ICONV Utility . . . . .	605
ICONVDEF Utility . . . . .	607
Format of the Translation Source File . . . . .	607
<b>Chapter 54. LOCALDEF Utility . . . . .</b>	<b>609</b>
Using LOCALDEF . . . . .	609
LOCALDEF Options . . . . .	610
/C (Continue If Errors) . . . . .	610
/F (Character Map) . . . . .	610
/I (Locale Source File) . . . . .	611

/W (Control Warnings) . . . . .	611
LOCALDEF Return Codes . . . . .	611

---

## **Part 11. Adding Application Resources . . . . . 613**

<b>Chapter 55. Using Resource Tools . . . . .</b>	<b>615</b>
IRC.EXE: The Resource Compiler . . . . .	615
ILINK.EXE: The Resource Linker . . . . .	616

<b>Chapter 56. An Overview of Working with Resources . . . . .</b>	<b>617</b>
What Is a Resource? . . . . .	617
Why You Should Link Resources to Your Applications . . . . .	619
Starting Resource Workshop . . . . .	619
Working with Resource Script Files . . . . .	620
Creating a Resource Script File . . . . .	620
Editing an Existing Resource Script File . . . . .	621
Adding Resource Scripts . . . . .	621
Generating Binary Resources from a Resource Script File . . . . .	627
Working with Graphic Files . . . . .	627
Creating a Graphic File . . . . .	627
Editing a Graphic File . . . . .	628
Customizing Resource Workshop . . . . .	628
Specifying Which Information Appears in the Resource Project Window . . . . .	628
Setting General Environment Options . . . . .	629

<b>Chapter 57. Working with Dialog Boxes . . . . .</b>	<b>631</b>
Editing a Dialog Box Resource Script . . . . .	631
Setting Dialog Box Properties . . . . .	632
Moving and Resizing a Dialog Box . . . . .	635
Adding a Standard Control to a Dialog Box . . . . .	636
Working with Custom Controls . . . . .	637
Selecting Controls . . . . .	637
Moving Controls . . . . .	638
Resizing Controls . . . . .	638
Fine-Tuning the Size of Controls . . . . .	639
Moving and Resizing Controls at the Same Time . . . . .	640
Grouping and Ungrouping Controls . . . . .	640
Setting the Tab Order of Controls . . . . .	640
Aligning Controls . . . . .	641
Setting the Windows Class of a Control . . . . .	644
Setting Common Properties using the Properties Window . . . . .	644
Setting Common Properties using the Style Window . . . . .	645
Setting the Unique Properties of a Control . . . . .	646



Testing a Dialog Box Resource . . . . .	653
Testing Two Dialog Boxes at the Same Time . . . . .	653
Returning to Edit Mode . . . . .	653
Customizing the Dialog Editor . . . . .	654
Using a Dialog Box Resource in an Application . . . . .	655
Programming a Dialog Box with the Windows API . . . . .	655
<b>Chapter 58. Working with Menus and Accelerator Tables . . . . .</b>	<b>657</b>
Editing a Menu Resource Script . . . . .	657
Adding a Menu or Separator to the Menu Bar . . . . .	658
Adding a Menu Item or Separator to a Menu . . . . .	660
Setting the Initial State of a Menu or Menu Item . . . . .	660
Changing the Identifier for a Menu Item . . . . .	661
Associating Help Text with a Menu Item . . . . .	661
Assigning an Accelerator to a Menu Item . . . . .	662
Copying, Moving, or Removing a Menu, Menu Item, or Separator . . . . .	663
Checking for Duplicate Identifiers . . . . .	663
Creating a Resource Script for a Floating Menu . . . . .	664
Customizing the Menu Editor . . . . .	664
Editing a Menuex Resource Script . . . . .	665
Editing an Accelerator Table Resource Script . . . . .	665
Adding an accelerator resource . . . . .	666
Changing the Identifier for an Accelerator Resource . . . . .	666
Changing the Keyboard Combination for an Accelerator Resource . . . . .	667
Copying, Moving, or Removing an Accelerator Resource . . . . .	667
Checking for Duplicate Keyboard Combinations . . . . .	668
Customizing the Accelerator Editor . . . . .	668
Using Menu and Accelerator Resources in an Applciation . . . . .	668
Programming menus and accelerators with the Windows API . . . . .	668
<b>Chapter 59. Working with String Tables . . . . .</b>	<b>669</b>
Editing a String Table Resource Script . . . . .	669
Adding a String Resource . . . . .	669
Changing the Characters of a String Resource . . . . .	670
Changing the Identifier for a String Resource . . . . .	670
Grouping String Resources to Maximize Memory Efficiency . . . . .	670
Copying, Moving, or Removing a String Resource . . . . .	671
Customizing the String Editor . . . . .	671
Using a String Resource in an Application . . . . .	671
<b>Chapter 60. Working with Graphics . . . . .</b>	<b>673</b>
Editing a Resource Script for a Graphic . . . . .	673
Choosing Colors . . . . .	674

Choosing a Pen Style . . . . .	675
Choosing a Brush Shape . . . . .	676
Choosing a Paint Pattern . . . . .	676
Drawing and Painting . . . . .	676
Drawing a Line . . . . .	677
Drawing a Shape . . . . .	677
Filling an Area with Color . . . . .	677
Adding Text . . . . .	678
Erasing an Area . . . . .	678
Selecting an Area . . . . .	679
Aligning an Area . . . . .	679
Moving or Resizing an Area . . . . .	679
Copying or Removing an Area . . . . .	680
Zooming in on or out from a Graphic . . . . .	680
Moving a Graphic Around in the Drawing Area . . . . .	680
Viewing an Icon or Cursor in Another Resolution . . . . .	681
Testing an Icon or Cursor . . . . .	681
Adding or Removing an Image from an Icon or Cursor . . . . .	681
Changing the Properties of a Bitmap . . . . .	682
Changing the Properties of a Cursor . . . . .	683
Changing the Properties of an Icon . . . . .	683
Changing the Properties of a Font . . . . .	683
Editing a Specific Character in a Font . . . . .	686
Customizing the Graphic editor . . . . .	686
Adding or Removing a Second Pane from the Edit Window . . . . .	686
Hiding or Showing the Tools and Colors Palettes . . . . .	687
Changing a Color on the Colors Palette . . . . .	687
Specifying Whether the Colors Palette is Saved . . . . .	688
Specifying Whether a Grid Appears When You Zoom in . . . . .	688
Using a Graphic Resource in an Application . . . . .	688
Programming a Bitmap with the Windows API . . . . .	688
Programming an Icon with the Windows API . . . . .	689
Programming a Cursor with the Windows API . . . . .	689
 <b>Chapter 61. Version Information and Custom Data Types . . . . .</b>	 691
Creating a Custom Data Type . . . . .	691
Editing a Resource Script for Version Information or a Custom Data Type . . . . .	691
Editing a Resource Script for Version Information . . . . .	692
Editing a Resource Script for a Custom Data Type . . . . .	698
Entering Data in a Resource Script . . . . .	699
Compiling a Resource Script . . . . .	700
Using a Version Information Resource in an Application . . . . .	700
Using a Resource for a Custom Data Type in an Application . . . . .	701

<b>Chapter 62. Help Table and Help Subtable Resources</b>	703
HELPTABLE Statement	703
Syntax	703
Example	704
HELPSUBTABLE Statement	704
Syntax	704
Example	705
Creating a Help Table Resource	705
Creating a Help Subtable Resource	705
Editing a Help Table or Help Subtable Resource	706

---

## **Part 12. Additional Utilities You May Find Useful** . . . . . 707

<b>Chapter 63. Program Maintenance Utility (NMAKE)</b>	709
Why Use NMAKE?	709
Running NMAKE	709
Using the Command Line	710
Using NMAKE Command Files	711
Options	712
Produce Error File (/X)	712
Build All Targets (/A)	713
Suppress Messages (/C)	713
Display Modification Dates (/D)	713
Override Environment Variables (/E)	713
Specify Description File (/F)	713
Display Help (/HELP or /?)	714
Ignore Exit Codes (/I)	714
Display Commands (/N)	714
Suppress Sign-On Banner (/NOLOGO)	714
Print Macro and Target Definitions (/P)	714
Return Exit Code (/Q)	714
Ignore TOOLS.INI File (/R)	715
Suppress Command Display (/S)	715
Change Target Modification Dates (/T)	715
Description Files	715
Description Blocks	715
Special Features	716
Targets In Several Description Blocks	717
Macros	718
Macros Example	719
Special Features	719
Macros in a Description File	719
Macros on the Command Line	719

Inherited Macros . . . . .	720
Defined Macros . . . . .	720
Macro Substitutions . . . . .	721
Special Macros . . . . .	721
Special Macros Examples . . . . .	722
File-Specification Parts . . . . .	723
Characters That Modify Special Macros . . . . .	724
Modified Special Macros Example . . . . .	724
Macro Precedence Rules . . . . .	724
Inference Rules . . . . .	725
Special Features . . . . .	726
Inference Rules Example . . . . .	726
Inference-Rule Path Specifications . . . . .	726
Predefined Inference Rules . . . . .	727
Directives . . . . .	727
Directives Example . . . . .	730
Pseudotargets . . . . .	730
Predefined Pseudotargets . . . . .	731
Inline Files . . . . .	732
Inline Files Example . . . . .	733
Escape Characters . . . . .	733
Characters That Modify Commands . . . . .	734
Turn Error Checking Off (-) . . . . .	734
Dash Command Modifier Examples . . . . .	734
Suppress Command Display (@) . . . . .	735
At Sign (@) Command Modifier Example . . . . .	735
Execute Command for Dependents (!) . . . . .	735
Exclamation Point (!) Command Modifier Examples . . . . .	735
EXTMAKE Syntax . . . . .	736
Macros and Inference Rules in TOOLS.INI . . . . .	737
TOOLS.INI Example . . . . .	737
 <b>Chapter 64. Using the Message Compiler . . . . .</b>	 739
MC Syntax . . . . .	739
Input Message File . . . . .	740
Header Section . . . . .	740
Message Definitions . . . . .	742
Language-Specific Message Text Definitions . . . . .	743
Input Message File Example . . . . .	745
Message Win32 API Calls . . . . .	746
Description . . . . .	746
Parameters . . . . .	746
Return Value . . . . .	749

<b>Chapter 65. Creating Online Documentation</b>	751
<b>Chapter 66. Demangling Compiled C++ Names with CPPFILT</b>	753
Using the CPPFILT Utility	753
Text Mode	753
Text Files	754
Output in Text Mode	754
Text Mode Options	754
/C (Class Names)	754
/H (Help)	755
/M (Symbol Map)	755
/Q (Do Not Display Logo)	755
/S (Special Symbol Names)	756
/T (Mangled and Demangled Names Together)	756
/Wnnn (Width)	756
Binary Mode	757
.obj and .lib Files	757
Output in Binary Mode	757
Binary Mode Options	758
/B (Operate in Binary Mode)	759
/H (Help)	759
/N (NONAME Keyword)	759
/O (Ordinals)	760
/P (Public Symbols)	760
/Q (Do Not Display Logo)	761
/R (Referenced Symbols)	761
/S (Special Symbol Names)	761
/U (Underscore Removal)	762
/X (Exported Symbols)	762
/Z (Suppresses Comments)	762
<b>Chapter 67. Resource Image Conversion Utility</b>	763
Starting the Utility	763
Using the Utility	764
Converting Your Images Using the Graphical Interface	764
Converting Your Images Using the Command Line Processor	765
<b>Glossary</b>	767
<b>Bibliography</b>	799
The IBM VisualAge for C++ Library	799
C and C++ Related Publications	799
Non-IBM Publications	799

<b>Index</b> . . . . .	801
------------------------	-----



## Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594, USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independent created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Canada Ltd., Department 071, 1150 Eglinton Avenue East, North York, Ontario M3C 1H7, Canada. Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

---

## Programming Interface Information

This book is intended to help you create programs using the VisualAge for C++ product. It primarily documents General-Use Programming Interface and Associated Guidance Information provided by VisualAge for C++ product.

General-Use programming interfaces allow the customer to write programs that obtain the services of the VisualAge for C++ compiler, debugger, browser, execution trace analyzer, and class libraries.

However, this book also documents Diagnosis, Modification, and Tuning Information. Diagnosis, Modification, and Tuning Information is provided to help you debug your programs.

**Warning:** Do not use this Diagnosis, Modification, and Tuning Information as a programming interface because it is subject to change.

Diagnosis, Modification, and Tuning Information is identified where it occurs by an introductory statement to a chapter or section.

---

## Trademarks and Service Marks

The following terms are trademarks of the International Business Machines Corporation in the United States or other countries or both:

DB2	PS/2
IBM	VisualAge
IBMLink	System Object Model
Open Class	SOM
Personal System/2	
WorkFrame	

Windows and Open Database Connectivity are trademarks of Microsoft Corporation.

INTERSOLV and DataDirect are trademarks of INTERSOLV, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Other company, product, or service names, which may be denoted by a double asterisk(\*\*), may be trademarks or service marks of others.

IBM's VisualAge products and services are not associated with or sponsored by Visual Edge Software, Ltd..





## About This Guide

This guide tells you how to use the IBM VisualAge for C++ for Windows, Version 3.5 product (referred to throughout the book as VisualAge for C++) to:

- develop and organize
- edit
- compile
- link
- debug
- analyze
- browse
- manage libraries for
- internationalize
- add application resources to

your C and C++ programs on the Windows NT and Windows 95 operating systems. For information on miscellaneous tasks not in the above list, see Part 12, “Additional Utilities You May Find Useful” on page 707.

For information on using the Visual Builder component of VisualAge for C++, see the *Visual Builder User's Guide*.

---

## Who Should Read This Guide

This guide is written for application and systems programmers who want to use VisualAge for C++ product to develop and run C or C++ applications. It assumes you have a working knowledge of the C or C++ programming language, the Windows operating system, and related products.

---

## How to Read Syntax Diagrams

This book uses two methods to show syntax. One is for commands, preprocessor directives, and statements; the other is for options.

### Syntax for Commands, Preprocessor Directives, and Statements

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

The ►— symbol indicates the beginning of a command, directive, or statement.

The —► symbol indicates that the command, directive, or statement syntax is continued on the next line.

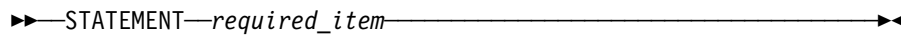
The  $\blacktriangleright$ — symbol indicates that a command, directive, or statement is continued from the previous line.

The — $\blacktriangleright$  symbol indicates the end of a command, directive, or statement.

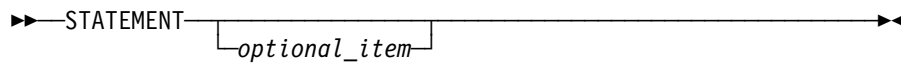
Diagrams of syntactical units other than complete commands, directives, or statements start with the  $\blacktriangleright$ — symbol and end with the — $\blacktriangleright$  symbol.

**Note:** In the following diagrams, STATEMENT represents a C or C++ command, directive, or statement.

- Required items appear on the horizontal line (the main path).

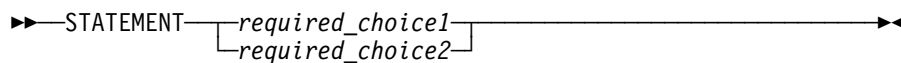


- Optional items appear below the main path.

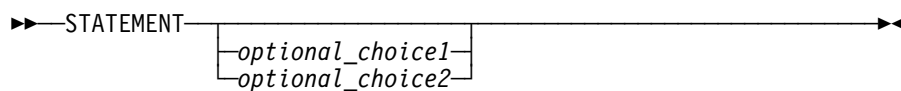


- If you can choose from two or more items, they appear vertically, in a stack.

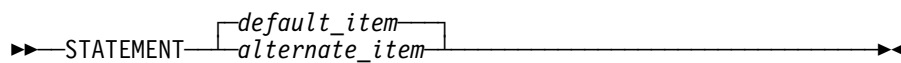
If you *must* choose one of the items, one item of the stack appears on the main path.



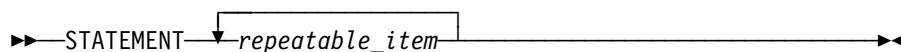
If the items are optional, the entire stack appears below the main path.



The item that is the default appears above the main path.



- An arrow returning to the left above the main line indicates an item that can be repeated.



A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.

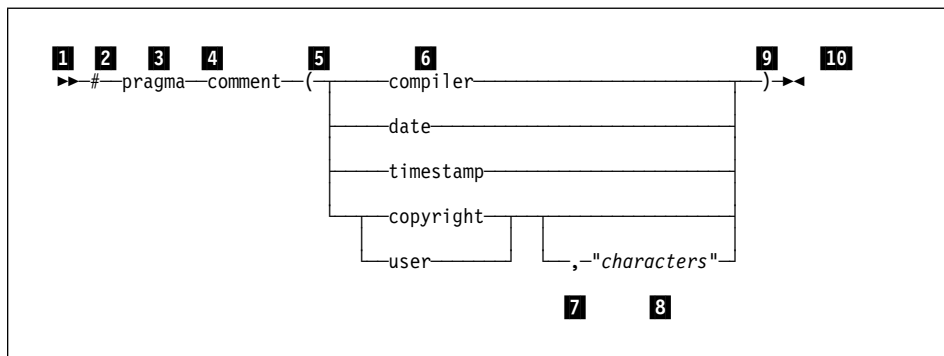
- Keywords appear in nonitalic letters and should be entered exactly as shown (for example, `pragma`).

Variables appear in italicized lowercase letters (for example, *identifier*). They represent user-supplied names or values.

- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

**Note:** The white space is not always required between tokens, but it is recommended that you include at least one blank between tokens unless specified otherwise.

The following syntax diagram example shows the syntax for the `#pragma comment` directive. (See the *Language Reference* for information on the `#pragma` directive.)



The syntax diagram is interpreted in the following manner:

- This is the start of the syntax diagram.
- The symbol `#` must appear first.
- The keyword `pragma` must appear following the `#` symbol.
- The keyword `comment` must appear following the keyword `pragma`.
- An opening parenthesis must be present.
- The comment type must be entered only as one of the types indicated: `compiler`, `date`, `timestamp`, `copyright`, or `user`.
- If the comment type is `copyright` or `user`, and an optional character string is following, a comma must be present after the comment type.
- A character string must follow the comma.
- A closing parenthesis is required.
- This is the end of the syntax diagram.

The following examples of the **#pragma** comment directive are syntactically correct according to the diagram shown above:

```
#pragma comment(date)
#pragma comment(user)
#pragma comment(copyright,"This text will appear in the module")
```

## Syntax for Compiler Options

- Optional elements are enclosed in square brackets [ ].
- When you have a list of items from which you can choose one, the logical OR symbol (|) separates the items.
- When you have a list of items from which you **must** choose one, the list of items is enclosed in angle brackets (< >).
- Variables appear in italicized lowercase letters (for example, *num*).

### Examples

#### Syntax      Possible Choices

/L[+|-]

/L  
/L+  
/L-

/Lt"string"    /Lt"Listing File for Program Test"

Note that, for options that use a plus (+) or minus (-) sign, if you do not specify a sign, the plus is assumed. For example, the /L and /L+ options are equivalent.

---

## How to Get Help

There are three kinds of online information available to you while you are using VisualAge for C++:

### Online documents

These are complete documents, like the one you are reading now, presented online. These documents contain detailed information on the different aspects of VisualAge for C++. For your convenience, the online documents are presented in:

- Standard format (.INF files). See “Getting Help Inside VisualAge for C++” on page xxxv for instructions on opening standard format documents from inside VisualAge for C++. See “Getting Help from the Command Line” on page xxxvi for instructions on opening standard format documents from the command line. For a list of the VisualAge for C++ documents that are available in standard format, see “Online Documents Available in VisualAge for C++” on page xxxvi.

### Contextual help

Contextual help is available throughout VisualAge for C++. This help tells you all about the elements that you see in the interface, including menus, entry fields, and pushbuttons.

### *How Do I* help

Many of the common tasks that you want to perform with VisualAge for C++ are described in *How Do I* help. The *How Do I* help for a task gives you step-by-step instructions for completing the task. There is overall *How Do I* help for VisualAge for C++, as well as individual task lists for each of its components.

## Getting Help Inside VisualAge for C++

All three kinds of help are available directly within the VisualAge for C++ interface:

- To get general contextual help for the component of VisualAge for C++ that you are using, press **F1** anywhere in the window.
- To get contextual help on a particular menu, menu item, or button, highlight the element and press **F1**.
- To get access to all of the help information that is available to you in a particular window, click on **Help** in the menu bar at the top of the window. This menu includes the following selections:
  - **Help Index**, an alphabetical list of all of the help topics that are available from this window
  - **General Help**, overall help for the window
  - **Using Help**, general information about the help facility
  - **How Do I...**, the *How Do I* help for the component
  - **Product Information**, a dialog that shows the level of VisualAge for C++ being used

In addition, there are selections that let you open all of online documents that are available in VisualAge for C++.

- To get detailed information, open the **Online Information** notebook in the VisualAge for C++ folder. In this notebook you will find tabs for **Guides**, **References**, and **How Do I** help. Each page in the notebook lists a variety of online documents that describe, in detail, the different aspects of VisualAge for C++. To open a particular online document, select the radio button for the document, and click on the **View** pushbutton.

## Getting Help from the Command Line

If you want, you can look at the online documents by issuing the `iview` command. The installation routine stores the online document files in the `\IBMCPW\HELP` directory. To view the *Language Reference*, for example, make `C:\IBMCPW\HELP` your current directory (substituting the drive where you installed VisualAge for C++ for `C:`) and enter the following command:

```
IVIEW CPPLNG.INF
```

If you want to get information on a specific topic, you can specify a word or a series of words after the file name. If the words appear in an entry in the table of contents or the index, the online document is opened to the associated section. For example, if you want to read the section on operator precedence in the *Language Reference*, you can enter the following command:

```
IVIEW CPPLNG.INF OPERATOR PRECEDENCE
```

## Getting Help for a Keyword or Construct

If you are editing a file using the Editor, you can get help for a keyword or construct by moving the cursor to the word and pressing **Ctrl+H**. In the other tools, you can get help for a keyword or construct by highlighting the word and pressing **Ctrl+H**.

## Online Documents Available in VisualAge for C++

The following documents are available in standard format:

<i>Building VisualAge for C++ Parts for Fun and Profit</i>	<i>Open Class Library Reference</i>
<i>C Library Reference</i>	<i>Open Class Library User's Guide</i>
<i>Editor Command Reference</i>	<i>Programming Guide</i>
<i>Frequently Asked Questions</i>	<i>SOM Programming Guide</i>
<i>Installation Guide and Product Overview</i>	<i>SOM Programming Reference</i>
<i>IPF User's Guide</i>	<i>User's Guide</i>
<i>IPF Programmer's Guide and Reference</i>	<i>Visual Builder User's Guide</i>
<i>Language Reference</i>	<i>Visual Builder Parts Reference</i>

---

## **Part 1. Developing with WorkFrame**

This part of the *User's Guide* explains how you can use the WorkFrame application development environment to manage your software projects.







## Chapter 1. Introducing WorkFrame

IBM WorkFrame is an application development environment that puts your tools at your fingertips. It simplifies the process of building and organizing software projects.

This chapter introduces you to WorkFrame.

---

### Overview

WorkFrame is a collection of objects and actions that organize your code into *projects*.

The project is the core of the WorkFrame environment. It encapsulates all the objects you need to build a single target. It also has an associated set of *actions* (like Edit, Compile, and Debug) that you can easily access.

You can organize complex applications into project hierarchies to give you a visual perspective of how your code is organized. You can perform a build from any point in a project hierarchy, giving you more control over the way you build your applications.

As you program, you will probably use tools, such as the visual builder, and performance analyzer. WorkFrame makes these tools available to you as context-sensitive actions in pop-up and pull-down menus in projects, and from the **Project** pull-down menus of many VisualAge for C++ tools.

You set options for actions quickly and easily using graphical user interfaces. When you invoke a **Compile** action from a pop-up or pull-down menu, WorkFrame invokes the compiler with your preset options. A window called **Build Smarts** lets you quickly modify options that affect the way your project is built. For example, you can quickly add debug and browse options so that you can easily build your project for debugging and production scenarios. **Build Smarts** also lets you set other options, such as generating a new makefile with every build, and whether or not subprojects should be built first.

## WorkFrame Introduction


---

### Opening a Project


There are two ways you can open a WorkFrame project to start working with it:

- Double-click on the WorkFrame icon in the VisualAge for C++ program group.
- Invoke WorkFrame from the command line by entering the command :

IWF [project]

where *project* is the name of the project file you want to work with. If you don't specify a file extension, then the extension `.iwp` is assumed.  See “Project Parts” on page 9 for more details about project files.

If you open a project from the command line without specifying a project file, or if you use the WorkFrame icon, the WorkFrame **initial dialog** is displayed. From the dialog, you can select **Open an existing project**, **Create a project**, or **Cancel**.

If you select **Open an existing project**, the open dialog appears where you specify the project file to open.  See “Creating Projects from Project Smarts” on page 17 for information about creating a project. Select **Cancel** to exit WorkFrame without opening a project.

---

### Working with Projects

Once a project is opened, the project view shows you all the source files for the project. The tools you need to work on your project are accessible from pop-up menus. To bring up a pop-up menu for a file, put the mouse pointer over a file, and then press mouse button two. You can also select a group of files, and put the mouse pointer over one of that group to display the pop-up menu relevant to that group.

## WorkFrame Introduction

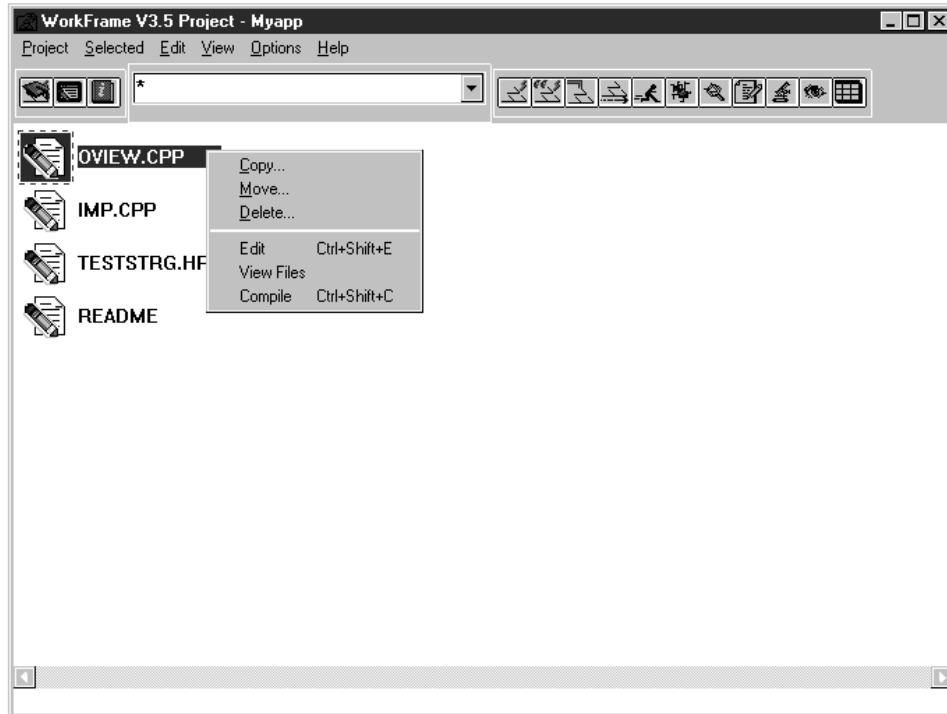


Figure 1. Project View with pop-up menu

In Figure 1, you see a pop-up menu containing the relevant actions for `oview.cpp` including **Compile** and **Edit**. The actions listed on a pop-up menu are context-sensitive to the type of file you are invoking the action on. These actions are called *file-scoped*.

Actions that apply to the entire project are available from the **Project** pull-down menu, pop-up menu or the toolbar. They are called *project-scoped* actions. The project pop-up menu is displayed by clicking mouse button two when the mouse pointer is in the project container but not over a file.

If you select to edit, or compile `oview.cpp`, the output from these actions is displayed in the monitor, which is an Editor command shell window. Double-click on compilation errors in the monitor to display an edit session of the source file with the line in error highlighted. Figure 2 on page 6 shows an edit session with a highlighted error for `oview.cpp`.

## WorkFrame Introduction

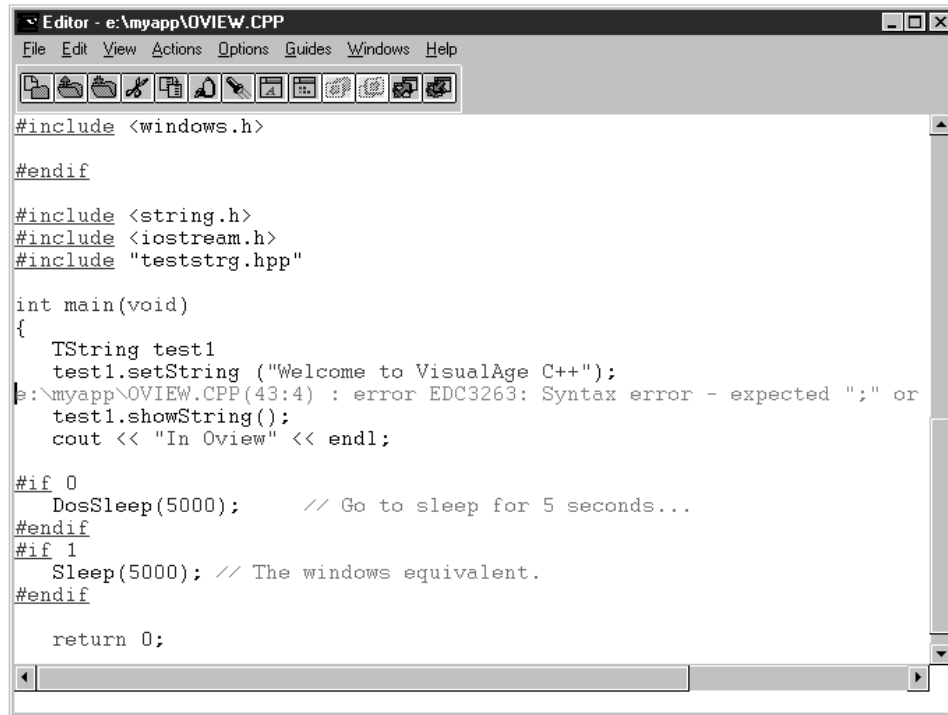


Figure 2. Error line highlighted in the editor

Correct the error, save the file, and then compile it again. There are two ways to recompile:

- Select **Compile** from the **Project** menu in the edit window.
- Returning to the project container, invoke the **Compile** action as you did the first time.

You will see the result of the second compilation in the same monitor window.

That little taste of working with WorkFrame projects gives you a sense of how useful this tool can be. The rest of this guide will explain, in detail, how to create and configure your own projects, set action options, organize project hierarchies, create make files, invoke builds, and more.

---

### Getting Help

You can get three kinds of online help while using WorkFrame:

- Contextual online help, which gives you help from within WorkFrame
- **How Do I?** help, which gives you step-by-step instructions on how to perform several project-related tasks
- An online version of this User's Guide, accessible from the Help pull-down menu.

### Using Contextual Help

Help on how to use any menu choice, window, or control is available through online contextual help. You can access it in one of the following ways:

- Select an item from the **Help** pull-down menu in any WorkFrame window. The Help menu in a project container gives you access to VisualAge for C++ manuals, **How Do I?** information, and contextual help for WorkFrame.
- Press **F1** from any WorkFrame window to get help on the current field.
- Highlight a menu item and press **F1** to get help on the menu item.
- Highlight the name of an action on a pop-up or pull-down menu and press **F1** for help on the action, where supported.
- Click on the **Help** pushbutton, where available.

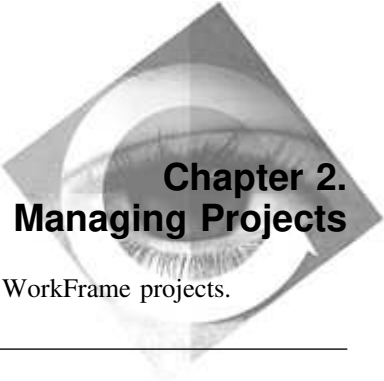
### Using How Do I? Information

Refer to **How Do I?** help when you need to accomplish a specific task, or when you want to explore WorkFrame functions.

You can access the **How Do I?** information in a number of ways:

- From the **Help** pull-down menu, select **How Do I?** or **How Do I?...Selections → WorkFrame**.
- Select the **Online Information** icon located in the main VisualAge for C++ program group on your desktop. Select WorkFrame from the **How Do I?** page of the **Online Information** notebook.
- Click on the **How Do I?** button on the left side of the project tool bar.

## **WorkFrame Introduction**



## Chapter 2. Managing Projects

This chapter explains how you can organize your own WorkFrame projects.


---


### Introducing Projects

The *project* represents the complete set of data and actions you need to build a single *target*, such as a dynamic link library (DLL) or executable file (EXE). It consists of a set of *project parts* and actions.

### Project Parts

*Project parts* are the data objects that make up the project. As well as a source file, a project part may also be a transient object, such as a target or intermediate object file created during the life of the project. A project part may also be another project. Including a project as a part of another project enables you to model hierarchies of projects.

A target is a specially designated project part. It is the result of a build action invoked on the project.  Builds are discussed in “The Build Utility” on page 38.

A project's parts are conceptually contained in the project; they are not physically stored in the project. They are stored in one or more directories called *source directories*. These are specified on the **Directories** page of the **Project Settings** notebook. All the files contained in the directories listed here are part of the project. The *working directory* is where any output files are created. When actions are executed, the working directory is their current directory.  Setting source and working directories are discussed in “Project Settings - Directories Page” on page 28.

WorkFrame stores the information about a project in two files, a *project file* and an *options file*. The *project file*, with the file extension *.iwp*, contains the general project information and the project settings. The *options file*, with the file extension *.iwo*, contains the options settings for the tools. These files must be in the same directory, and they must have the same filename. They do not need to be in a WorkFrame source directory. When you start WorkFrame from the command line, you specify the name of the *project file*.

### Actions

An *action* is a description of a tool or a function of a tool that can be used to manipulate a project's parts, or participate in a build. Some actions, such as compile, have options that can be changed using the option dialogs available on the **Options** pull-down menu.

## Managing Projects

---

### Project Views

A project has two views:

- Icon view
- Tree view

**Icon view** and **tree view** are discussed in this chapter. By default, a project opens to its icon view, but you can select the tree view from the **View** menu.

### Icon View

Figure 3 shows a fully expanded Icon view of a project.

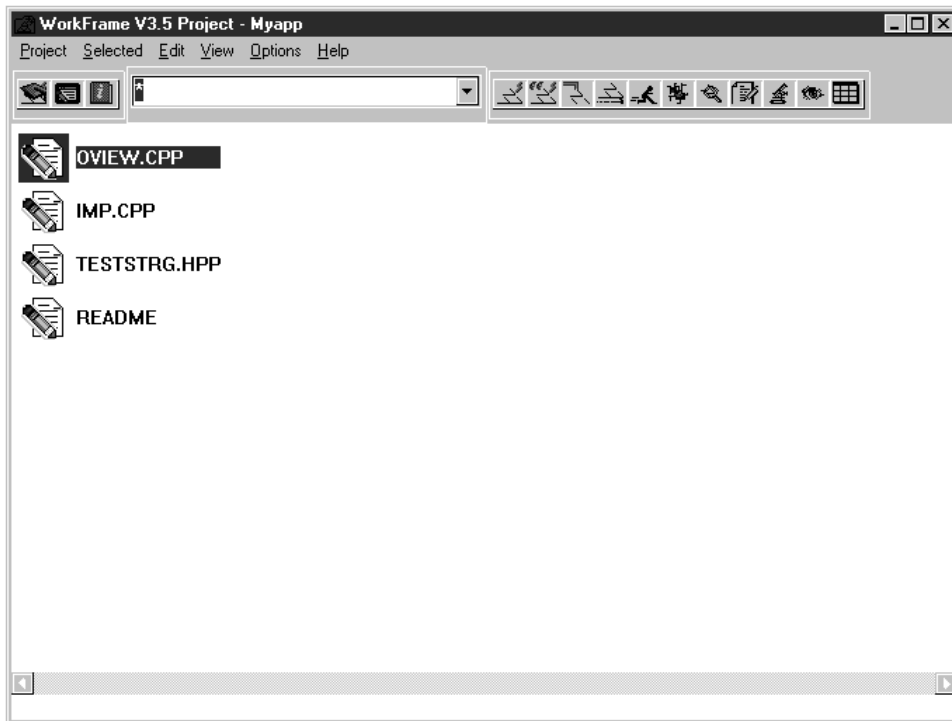


Figure 3. Project - Icon View



## Managing Projects

A project container has the following controls:

### System menu


The system menu contains menu items for manipulating the window that WorkFrame is displayed in.

### Project toolbar

The project toolbar has three fixed buttons on the left-hand side:



#### Build Smarts

Displays the **Build Smarts** window where you can set build features for your VisualAge for C++ projects.  See “Build Smarts” on page 34 for more information on how to use Build Smarts to aid you in your project builds.



#### Monitor

Displays the project **Monitor** window where the output of monitored actions is displayed. For example, compiler and linker output is displayed in this window.



#### How Do I?

Displays the online **How Do I?** help for WorkFrame. It contains instructions for how to accomplish common WorkFrame tasks.

The right-hand side of the project toolbar also contains buttons for launching frequently-used, project-scoped actions like Build, Debug, and Run. Each button represents a single project-scoped action.

**Performance Note:** All WorkFrame toolbars have hover help. Hover help is a balloon that appears whenever you position the mouse pointer over a toolbar button. It contains help text about a button. You can turn off hover help when you become familiar with the toolbar buttons by deselecting the **Toolbars → Hover help** menu item from the **View** menu. You will see some performance gain when you do this.

## Managing Projects

### Menu bar


The project menu bar contains menu items to launch project- and file-scoped actions, bring up the project's **Settings** notebook, set action options, and get help for WorkFrame projects.

The **Project** menu lists all the project-scoped actions available to the project. The menu also contains actions to open or create another project, to close the project view, and to exit WorkFrame.

The **Selected** menu contains the actions that apply to any selected project parts. The menu changes depending on the type of the parts selected. **Move**, **Copy**, and **Delete** are menu items for every part type.

Note that project- and file-scoped actions are also accessible from pop-up menus.

The **View** menu contains controls for opening different views on the project, including the project's **Settings** notebook. It also contains view options for the project toolbar and information area.

The **Options** menu is a list of actions whose options you may need to change. When you select an action from this menu, the options dialog for the action appears. The **Options** menu also contains a menu item for the **Build Smarts** window that lets you quickly change options for actions used in a project build.  See “Tools Options” on page 32 and “Build Smarts” on page 34 for more information about setting action options.

The **Help** menu contains items for referencing WorkFrame online help, including WorkFrame **How Do I?** help and other online VisualAge for C++ manuals.


## Managing Projects

### Parts filter

The parts filter is an entry field on the project toolbar where you can enter a file mask to filter the view of project parts. Only the parts that match the specified mask are displayed in the parts container. You can also use the drop-down list box to select from the masks that are available to the project. To use multiple file masks, separate them with a semicolon.

The current setting of the parts filter is only saved if you press the Enter key to activate it. It is saved for all the project views.

### Parts container

This is the place where the project parts are displayed. All files from the WorkFrame source directories are considered project parts. Source directories are specified in the **Directories** page of the project's **Settings** notebook.  See “Project Settings - Directories Page” on page 28 for more information on how to specify a project's files.

Project parts are displayed as icons in the project's icon view container.


### Information Line

Explanations of the toolbar buttons are displayed here as the mouse moves over them when hover help is enabled.

## Managing Projects

### Tree View

In a project tree view, the project container is organized into a hierarchy of files from each directory. The basic tree consists of the project's source directories. Each can be expanded to display the files they contain.

All controls for a project's Tree view are identical to those in its Icon view.  See “Icon View” on page 10 for more information on the controls.

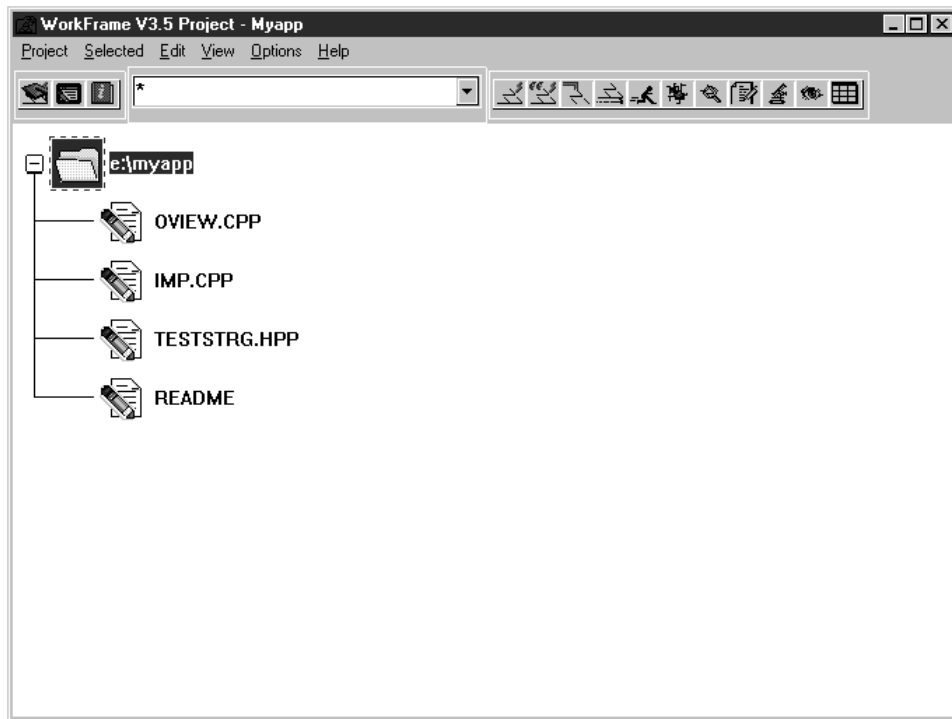



Figure 4. VisualAge for C++ Project - Tree View

---


### Creating a Project

There are two ways you can create a WorkFrame project:

- Use Project Smarts.

Project Smarts generates a complete project with skeletal code for an application type that you choose. The various kinds of applications that you can choose from include User Interface Class Library and Visual Builder. For example, if you choose the UI Class Library application from Project Smarts, a project is created on your desktop that has a completely configured project environment and template code for an UI Class Library program.  See “Creating Projects from Project Smarts” on page 17 for more information on how to create projects.

- Copy another project's files.

Copy and rename the project and option files for a similar project.  See “Project Parts” on page 9 for more information on how projects are represented in the file system. You can share source directories with the project being copied from, or use new source directories. If you are sharing source directories, use a different working directory so one project's build results are not overwritten by the others.



If you have no existing source files to create a project with, and you have a specific kind of target in mind, then Project Smarts is the best way for you to create a project. If you already have source files to work with, you can either copy a project that is similar to your application such as a sample, or create a default project from **Project Smarts**. A default project has no generated template files. Only the project file is created.

## Managing Projects

### Project Smarts

Project Smarts is a powerful tool to help you quickly get started writing VisualAge for C++ applications. It is a catalog of skeleton applications you can use as a base with which to write your own applications. It contains projects of common applications including:

- Visual Builder Application
- UI Class Library Application
- Data Access Application
- Direct-to-SOM Application
- Resource Dynamic Link Library
- C++ Dynamic Link Library
- C Dynamic Link Library
- Compound Document Framework
- Standard Hello World
- Collection Class Library
- IPF Document or Help
- RTF Help

When you instantiate one of these Project Smarts applications, a fully-configured, development-ready project is created on your desktop. All the actions, options, and environment variables you need to develop a similar application are preconfigured for you. Each project is created with template source files to help you get started quickly on the real work, without having to set up the basics every time. Project Smarts applications are skeleton programs that you can actually build and run.

A Project Smarts application is distinct from a sample project; it does not teach you programming techniques or concepts. The code provides a starting point for you to build on when developing your own applications from the code templates.

## Managing Projects

### Creating Projects from Project Smarts

There are three ways to start Project Smarts:

- Select **Create new project** from the **Project** pull-down menu.
- Select **Create new project** from the WorkFrame initial dialog. The initial dialog is displayed when you double-click on the WorkFrame icon in the VisualAge for C++ program group, or when invoke WorkFrame from the command line without specifying a project file.
- Project Smarts can also be started from the command line. To invoke Project Smarts from the command line, type ISMARTS.

The **Project** page of the Project Smarts notebook is displayed. The **Project types** list box in the notebook contains a list of VisualAge for C++ project types. As you select a project title, the **Description** field is updated with a short description of the project.

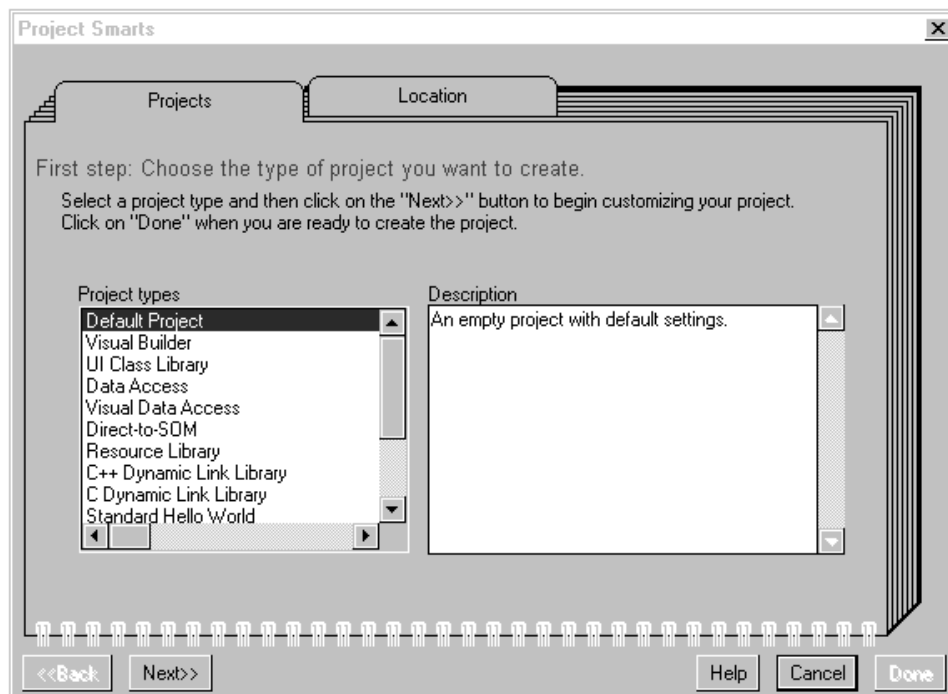


Figure 5. VisualAge for C++ Project Smarts catalog

## Managing Projects

To create one of the Project Smarts projects, select its name and then click on the **Next** push button. Each application is different, so the pages displayed will be different. Each page is labeled with a step number, and you can return to the previous page by clicking on the **Back** push button.

The **Location** page is a step that applies to all applications. Use it to specify the name of the project file, and the directory where the project file should be created. You also specify the source directories where the generated template files are stored. In addition to these fields, the project name is also entered on the **Location** page.

After the project has been created, it is opened. You are now ready to begin working on developing the specifics of your application.



---

### Organizing Projects

Organizing your projects well is the key to using the WorkFrame environment effectively. The organization of your projects determines the way in which they are built, and the way in which makefiles are generated for your applications.

Most of your applications will most likely consist of a hierarchy of projects, rather than a single project, unless they are very simple. It is important that your project hierarchy reflects the project targets and dependencies between components for builds to be performed correctly. Follow these guidelines for a well-defined project hierarchy:

1. Create a separate project for each target or build path in your application. A *target* is defined as a single part or file that the project will build, such as an executable file, dynamic link library, or help file. A *build path* is a sequential processing of actions, with no conditional or alternative routes, that produces a single target file. Intermediate files may also be produced during the processing of a build.



If your application consists of two DLLs, a LIB, an EXE, and a HLP file, for example, you should create a separate project for each DLL, the EXE, the LIB, and the HLP file. If LIBs are also produced from the DLLs, the DLL and the LIB can be part of the same project. The DLL would be the designated target of the project, while the LIB is considered an intermediate file. In this example, you would create five projects in all.



Figure 6. Five projects in a sample application

2. After you have created the projects, determine the dependencies that exist between them.



For example, the DLLs may depend on the LIB, and the EXE depends on the DLLs. Also determine the project that is at the root of the dependency hierarchy, that is, the project that has no projects dependent on it. In this example, it is the EXE.

## Managing Projects

3. Nest the projects to reflect the dependency tree. That is, if a project *EXE* depends on another project *DLL*, place project *DLL* into project *EXE*. Project *DLL* is then said to be *nested* within project *EXE*, and is a subproject of project *EXE*. A project is nested in another project when its project and option files are located in one of the source directories of the nesting project. To nest project *DLL* into project *EXE*, the project and options files for project *DLL*, *dll.iwp* and *dll.iwo*, would be moved into one of the source directories of project *EXE*.

In a situation where two or more projects (say *First DLL/LIB* and *Second DLL/LIB*) depend on the same project *LIB*, nest one of the two depending projects within the other (say, nest *Second DLL/LIB* within *First DLL/LIB*), and then nest the mutual dependency project *LIB* within project *Second DLL/LIB*.

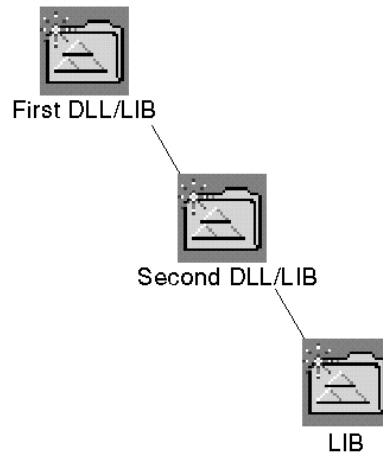


Figure 7. Projects *First DLL/LIB* and *Second DLL/LIB* depend on project *LIB*

The root project in the dependency tree will directly or indirectly contain all the other projects in the application.

## Managing Projects



Consider the example application that consists of an EXE, two DLLs, a LIB, and a HLP file. The two DLLs depend on the LIB, the EXE depends on the DLLs and the HLP file. The LIB and HLP files do not depend on any other targets. The project hierarchy would look like the one represented in Figure 8.

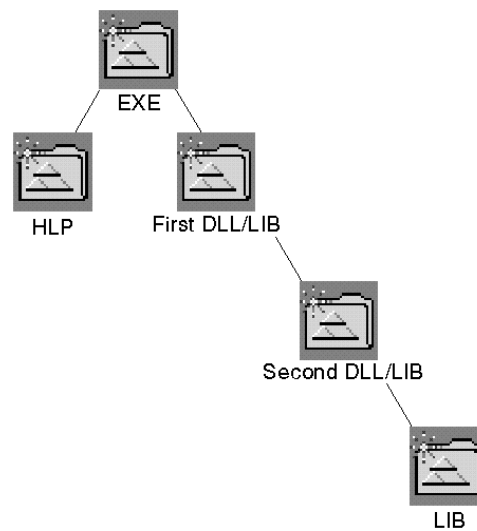



Figure 8. Project hierarchy of a sample application

The *EXE* project is the root project that contains all the other projects. *First DLL/LIB* and *Second DLL/LIB* both depend on the *LIB*, so one nests the other before nesting the *LIB* project. A LIB file is also produced by each of the two DLL builds.

You should nest projects this way because the WorkFrame Build utility infers the build sequence from the project hierarchy. When you use the Build utility to build a project, you have the option of having it build descendant projects first. Following this scheme ensures that a project's descendants contain the current project's dependencies.  See “The Build Utility” on page 38 for more details.

## Managing Projects


4. All the files used by the projects in your application can be stored in one or more directories. For example, all the source files to build the DLLs and LIBs could be stored in one directory, all the source files for the EXE in another, and all the HLP files in a third or fourth directory.

**Nesting Note:** If you store the files for each project in a different directory, the target of a nested project is not automatically considered a part of the nesting project. You must include the directory that contains the required target in the source directories list of the nesting project. As a better alternative, you could define an environment variable in the nesting project so that the required target is available. For example, the *EXE* project could define the *LIB* environment variable to contain the working directory path of the *LIB* project whose target it depends on.



For example, if the *First DLL/LIB* project's files are stored in a different directory from the *EXE* project, say in D:\MYAPP\LIB1, the *EXE* project must define the *LIB* environment variable to contain that directory so that the Link action is able to find the required library file:

```
LIB=D:\MYAPP\DLL1;%LIB%
```

 See “Project Settings - Environment page” on page 30 for information on setting environment variables.



Your project's directory structure does not have to mimic your project organization. A project can contain source files that are stored in multiple directories. For example, two projects that build two different targets from the same source files can share one or more source directories, but have separate working directories.

If your project has any header files that are stored in directories other than the project's working directory, you should define the *INCLUDE* environment variable to reference the header files.

### Project Geometry

To put things into perspective, this section discusses two different, but simultaneous structures that a project can maintain: one conceptual and one physical.

#### Nesting

A nesting graph of all the projects in your system would be in the form of a tree, and would connect lines between projects and their subprojects. The organization of a project expresses the interdependency relationships that exist between itself and its descendants. This kind of project structure is very important for setting up project builds using the WorkFrame Build utility.

#### Directories

The source directories of a project determine a project's parts. The source directories of a project can mimic its nesting structure, or serve a different organizational purpose, such as grouping files by type. For example, private and public header files could be stored in separate directories from the rest of the source.

## Managing Projects



## Chapter 3. Project and Tool Option Setup

This chapter introduces you to setting up your projects and tool options.

---

### Project Settings

A project has a number of settings, such as the source and working directories, that you can modify. This section will take you through all the settings in a project **Settings** notebook.

To open the **Settings** notebook, you select a **Settings** notebook page from the project's **View** menu **Settings** cascade.

The pages of the **Settings** notebook all have four buttons. Changes made to fields in the **Settings** notebook are saved and the notebook is closed when the **OK** button is used. The **Cancel** button returns all fields on all pages to the state they were in before the notebook was opened, and closes the notebook. The **Undo** button returns all fields on the page to the state they were in when the notebook was opened. The **Help** button provides help for the fields on the page.

## Project and Tool Options Setup

### Project Settings - Target Page

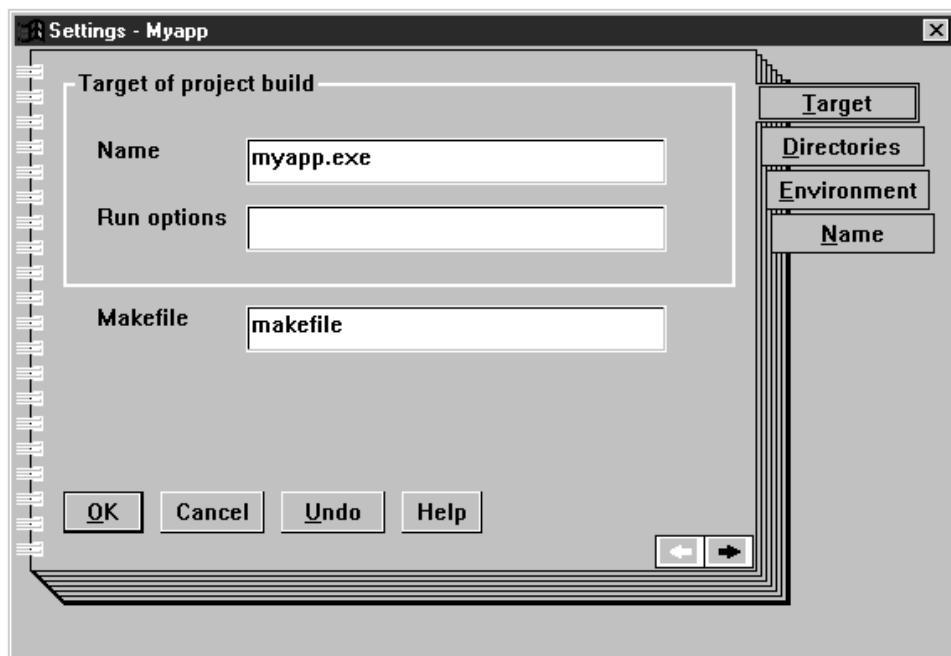


Figure 9. Project Settings - Target page

You can set the following information having to do with a project's designated target in the **Target** page of the project's **Settings** notebook:

#### Target of project build

Specify the target's name and parameters.

##### Name

The name of the project's target, for example, PROGRAM.EXE.

##### Target Notes:

1. A project can build more than one target, but WorkFrame recognizes only one project part as the designated target of the project. See “The Build Utility” on page 38 for more information on building a project's target.
2. The object you specify as the project's designated target does not have to exist. It is understood to be the target of a build action on the project.




## Project and Tool Options Setup

### Run options

The parameters entered here are passed to the target when it is executed. If the target is not an executable file, the contents of this field are ignored.

### Makefile

The name of the project's designated makefile that is used when a project-scoped build action is invoked. (  See “The Build Utility” on page 38 for more information about builds.) A project can have more than one makefile, for example, one that builds the target with debug information and another that builds it optimized, but WorkFrame recognizes only one as the designated makefile of the project. Makefiles are recognized by having either the name makefile or the .mak file extension.

## Project and Tool Options Setup

### Project Settings - Directories Page

You specify the source directories for your project in the **Directories** page. All the files in the source directories are assumed to be the project's parts.



Figure 10. Project Settings - Directories page

On the **Directories** page, you specify the following settings:

#### Source directories for project files

List the source directories where project files are to be stored, one path name per line. If the directories do not exist, you are prompted for permission to create them. If files already exist in one or more of the specified directories, they automatically become project parts. For example, you could enter the path names:


```
D:\MYAPP\DLL  
D:\MYAPP\HEADERS
```

if the files are to be located in these directories.

#### Working directory

Select the working directory for your project from the directories you specified in the **Source directories** field. This directory is used:

## Project and Tool Options Setup

- To store any makefiles created by the MakeMake and Build utilities (  described in “The MakeMake Utility” on page 52 and “The Build Utility” on page 38).
- As the current directory from which actions are launched. Because many tools, such as the VisualAge for C++ icc compiler, place their output in the current directory, tool output such as object files are often stored here.

By default, the first directory in the list of source directories is the project's working directory.

## Project and Tool Options Setup

### Project Settings - Environment page

On the **Environment** page, you specify the environment variables you wish to set, one variable per line. Enter in the format *name=value*, where *name* is the name of the environment variable, and *value* is the value you want it set to.

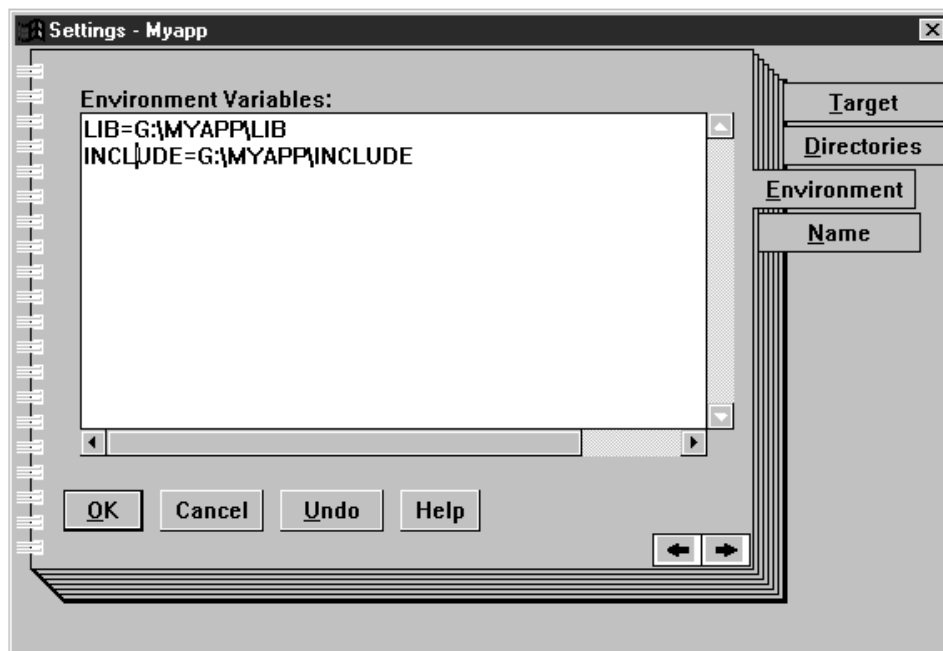


Figure 11. Project Settings - Environment page

The environment variables defined are active only for your project, without affecting variables in other sessions. The environment variables listed in a project's environment are set for any tool launched as an action from a WorkFrame project.

## Project and Tool Options Setup

### Project Settings - Name Page

You specify the name of your project on the **Name** page. The project name appears in the project window's title bar.

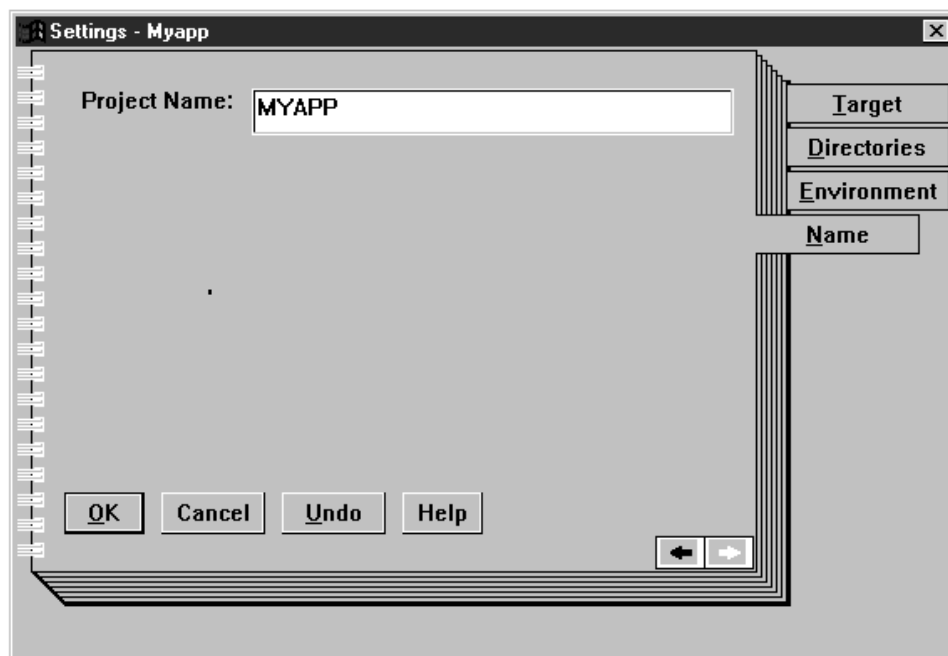


Figure 12. Project Settings - Name page

## Project and Tool Options Setup

---

### Tools Options Setup

WorkFrame tools options also have default settings. These settings can be modified by using the option dialogs on the **Options** pull-down menu.

### Tools Options

Each tool used in a makefile has a set of options associated with it. The options represent the parameters passed to the tool to configure its behavior when its action is invoked. For example, to have the compiler generate Browser information, you need to invoke the compiler with the `/Fb` option.

When you use WorkFrame projects to organize your tools and code, you can set these options once, and then when you invoke the action, by itself or as part of a Build, the options are always set for you. If the tool you are using provides customized WorkFrame support, you won't need to remember command-line options like `/Ge-` that tells the compiler that you're building a DLL, or `/Stnumber` to specify the stack size to the linker. Instead, you set options in a graphical user interface, like a dialog or notebook, with full access to online help. Figure 13 on page 33 shows you an image of the compile options dialog.

## Project and Tool Options Setup

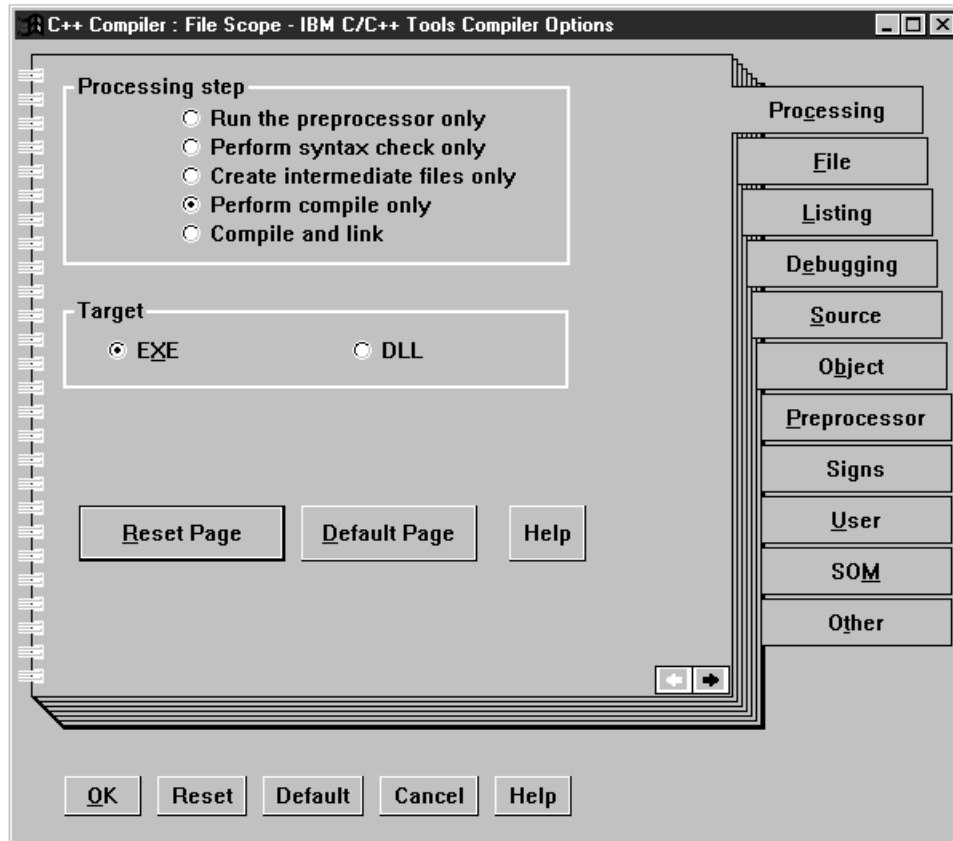


Figure 13. The VisualAge for C++ Compiler options dialog

To set options for an action, select its action name from the **Options** menu on the project menu bar. Only some actions are available on the **Options** menu.

## Project and Tool Options Setup

### Build Smarts

Build Smarts is a fast path for setting options for VisualAge for C++ actions that affect the way your project target is built. The options that you set in the **Build Smarts** window work in combination with the options that are set for the individual actions themselves. By default, Build Smarts is enabled.

To display the **Build Smarts** window, select **Build Smarts** from the **Options** menu, or from the toolbar. The **Build Smarts** window is illustrated in Figure 14 on page 35.



## Project and Tool Options Setup

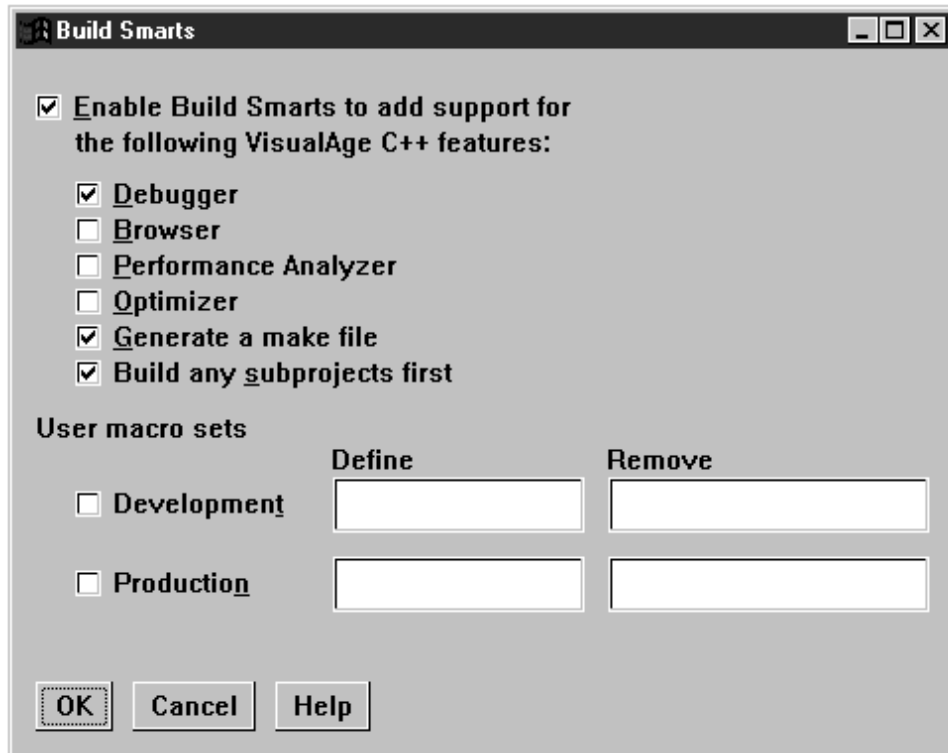


Figure 14. Build Smarts window

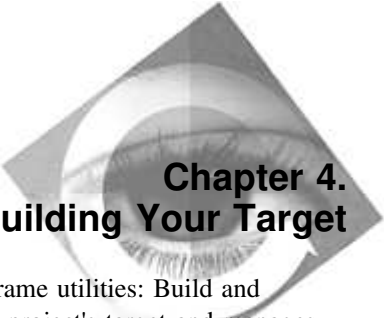
Each setting in the **Build Smarts** window can affect one or more VisualAge for C++ actions. For instance, the **Debugger** check box affects the VisualAge for C++ Compiler and Linker options. Selecting the **Debugger** check box effectively adds the `/Ti` option to the VisualAge for C++ Compiler command line, and `/DE` to the VisualAge for C++ Linker command line, when you initiate a build involving those actions.

## Project and Tool Options Setup

The Build Smarts settings do not change the options already set for the individual actions in their options dialogs. The tools setup settings are simply added to what is already set for the involved actions. When there are conflicts, the Build Smarts settings override those already set for the individual actions.

Two important build options are also included here: **Generate a makefile** and **Build any subprojects first**. Select the first option to generate a new makefile as part of running the build. Select the second option to build any child projects before building the current project so that all the dependencies are up to date.

At any time, you can disable the Build Smarts options by deselecting the **Enable Build Smarts** features check box if you only want to use the options set for the individual VisualAge for C++ actions.



## Chapter 4. Building Your Target

This chapter introduces you to two very useful WorkFrame utilities: Build and MakeMake. The Build utility dynamically builds your project's target and manages your makefile for you. MakeMake is WorkFrame's makefile creation utility.

---

### Build and Make

A build action and a make action are very similar. Both are project-scoped actions used to build the target of a project. There are, however, very important differences between the two:

#### Make

The make action runs a make utility, like NMAKE, against an existing makefile. The makefile is typically a project part that was generated by the WorkFrame makefile generation utility, MakeMake, or written by hand. A makefile is a static object that reflects the project settings at the time the makefile was generated. Whenever changes are made to the project settings, such as the action options, you must update or regenerate the makefile to reflect the most current project settings.

Makefiles are useful when you want to package the source files for distribution, and when you want to build the target in a constant and predictable manner. A project may contain multiple makefiles to build various forms of the same target, for debugging or profiling, and optimization, for example. However the project recognizes only one makefile as the object of a project-scoped make action. A file-scoped make action can be invoked on any makefile in the project. Makefiles are recognized by having either the name `makefile` or the `.mak` file extension.

#### Build

A build action runs the WorkFrame Build utility, which also uses a makefile and make utility to build the project target. However, the Build utility can dynamically generate the makefile each time a build is initiated. Therefore, if the project settings change, the build values are implicitly updated. You don't have to do anything to update the makefile beyond changing the project settings.

The build utility also manages dependencies between projects in a project hierarchy. It can build the projects lowest in the hierarchy before building the ones higher up. You can initiate a build anywhere within a project hierarchy.

## Building Your Target

The Build utility also provides two additional conveniences: it can lock projects to prevent concurrent builds from colliding, and validate that the build target is not in use before the build is started.



---

## The Build Utility

The WorkFrame Build utility eliminates the need for explicitly generating and maintaining makefiles. It uses the MakeMake utility to generate a new makefile each time a build is initiated. To build only out-of-date files, select **Build normal** from either the project toolbar or the **Project** pull-down menu. You can build all targets, even if they are not out of date with respect to their dependent files by selecting **Rebuild all** from either the project toolbar or the **Project** pull-down menu.

## Build Prerequisites

The Build utility relies on a well-defined project setup to correctly generate the build rules and dependency information for your project:

- The project dependencies must be expressed within the project hierarchy.  Follow the guidelines in “Organizing Projects” on page 19.
- Correct build options, especially the build actions, must be set as described in  “Setting Build Options” on page 39 .

## Building Your Target

### Setting Build Options

You set build options for a project the same way you set options for any other action. From the **Options** pull-down menu, select **Build normal** or **Rebuild all** to display the **Build** options notebook.

The build options you set in the **Build** options notebook can be overridden by **Build Smarts** settings. See “Build Smarts” on page 34 for more information about **Build Smarts**.

The **Build** options notebook for either action has the following pages:

- Actions
- Make
- Project
- Display

The **Build** options notebook has five buttons:

- The **OK** button saves the updates and then closes the notebook.
- Selecting the **Reset** button will reset the fields on all pages in the notebook back to the same values as when the notebook was opened.
- The **Default** button resets the fields on all pages in the notebook to their default values.
- The **Cancel** button closes the notebook without saving any updates to the fields in the notebook.
- The **Help** button provides help for the notebook.

Each page in the **Build** options notebook has a **Help** button that provides help for the fields on the page. The notebook pages also have a **Reset** button to set the values of the fields back what they were when the notebook was opened. A **Default** button, which resets the fields on the page to the default values, is provided on each page as well.

## Building Your Target

### Build Options - Actions Page

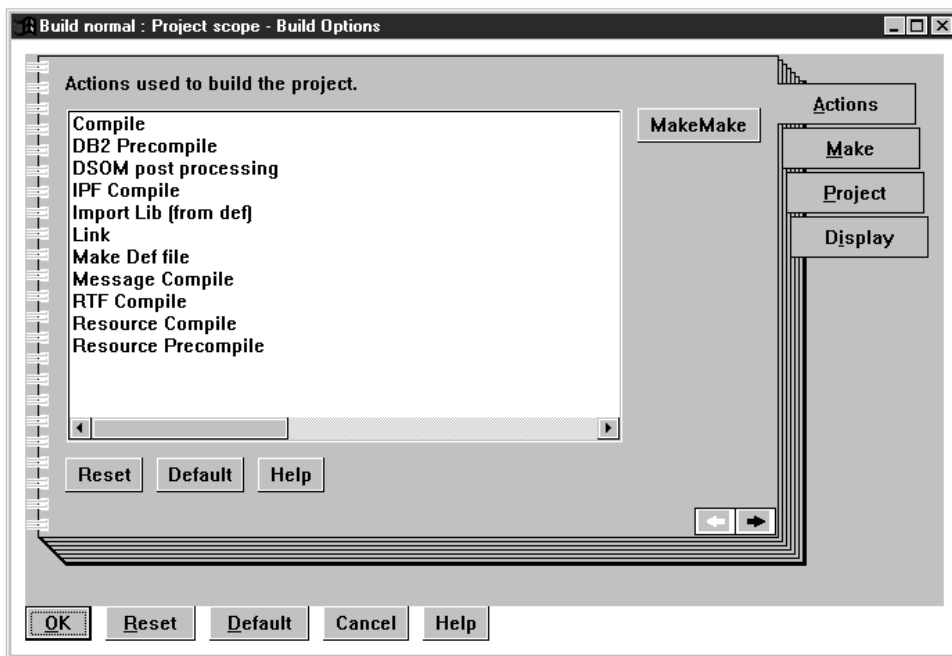


Figure 15. Build options - Actions page

The most important information needed to build a project's target is the set of actions needed to build it. You select these actions from the **Actions** page of the **Build** options notebook.

The **Actions** list box contains a list of file-scoped actions eligible for participation in a build. The Build utility attempts to formulate the build rules for your project by examining the source files and the source and target types of file-scoped actions in the project.

## Building Your Target

You can select the build actions from one of two places: from the **Actions** list box on the **Actions** page, or click on the **MakeMake** button to invoke the MakeMake utility, where you can select actions and explicitly create a makefile.

Where you select the Build actions depends on:

- Whether you want to have descendant projects use the same set of Build actions. If you do, you will need to:
  1. Select the Build actions from the **Actions** list box.
  2. Select the **Pass Build settings to subprojects** on the **Project** page.
  3. In the Build options for the descendant projects, select the **Use build settings from parent project** on the **Project** page (this is the default).

The build actions you select in the **Actions** list box apply to *all* the source files in the project, based on their type. If you want the actions to only apply to some of the source files in the project, you must select the actions and applicable source files from MakeMake.

- Whether you want to explicitly select source files to which the build actions should apply. If you do, you will need to select the build actions from MakeMake. However, descendant projects cannot use the current project's build actions if they are specified from MakeMake.

MakeMake saves the actions from your last successful makefile generation. If you already have a makefile generated by MakeMake, you can use the previously saved actions by deselecting any actions selected in the **Actions** list box.

**Note:** If any actions in the **Actions** list box are selected, those actions are used for the build, even if build actions were previously set from MakeMake. Deselect any selected actions in the **Actions** list box if you want the settings from MakeMake to apply to your build.

## Building Your Target

### Build Options - Make Page

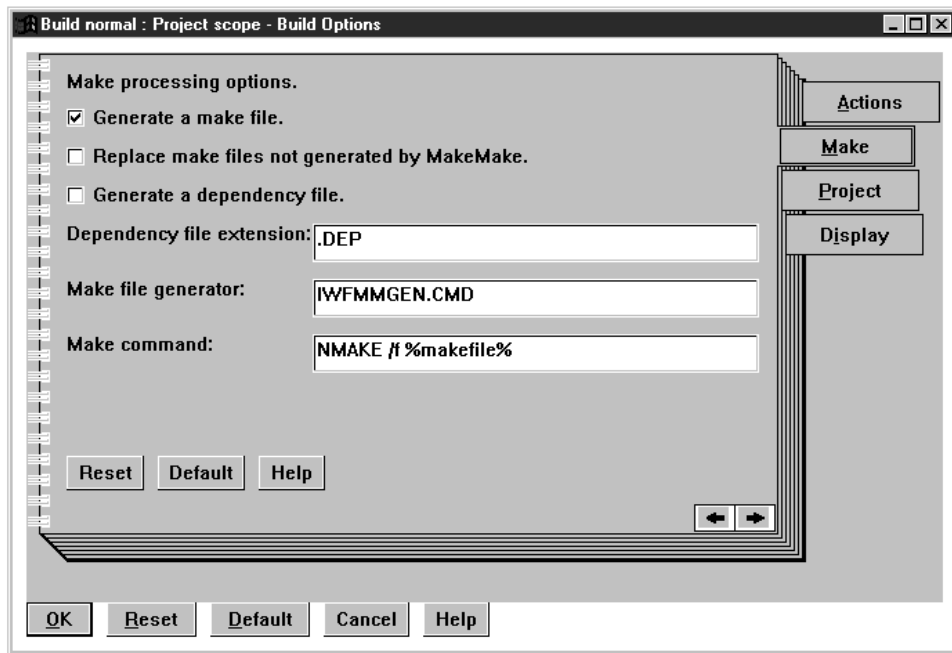


Figure 16. Build options - Make page

The options in the **Make** page relate to the make utility, and how the makefile is managed and maintained.

#### Make processing options

Select whether or not the makefile and separate dependency file are to be generated before each build. Generating these files before each build guarantees that the makefile used to build the project target is up to date with the latest project and action settings.

##### Generate a makefile

Select this option if you want the makefile generated before each build. If you do not select this option, the existing makefile is used.

##### Replace makefiles not generated by MakeMake

Select this option if you want to replace makefiles not generated by MakeMake. This includes makefiles that you may have edited after MakeMake generated them.



## Building Your Target

### Generate a dependency file

Select this option to create a separate dependencies file before every build. If you select the **Generate a makefile** option without selecting this option, then the dependencies are stored with the makefile.

### Dependency file extension

Enter the filename extension for the dependency file, if one is to be generated. The default is .dep.

Selecting both the **Generate a makefile** and **Generate a dependency file** options frees you from having to maintain the make and dependency files. If you do not select either of the options, the makefile and dependency file will not be generated or updated. You can update the makefile and dependency file manually by running MakeMake.

### Make utility

Use these fields to identify the make utility and options you want to use to build the target.

### Makefile generator

The makefile generator is a script used by MakeMake to generate the makefile in a format understood by the make utility. The default script is IWFMMGEN, supplied by WorkFrame. This script enables MakeMake to generate makefiles that are compatible with the NMAKE make utility included with VisualAge for C++.

### Make command

This field contains the command line to execute the make utility, including the required options. The default for the **Build Normal** option is to use the NMAKE utility with the command line `NMAKE /f %makefile%`, which invokes the NMAKE utility on the project's designated makefile. including the required options. The default for the **Rebuild all** option is to use the NMAKE utility with the command line `NMAKE /a /f %makefile%`.

## Building Your Target

### Build Options - Project Page

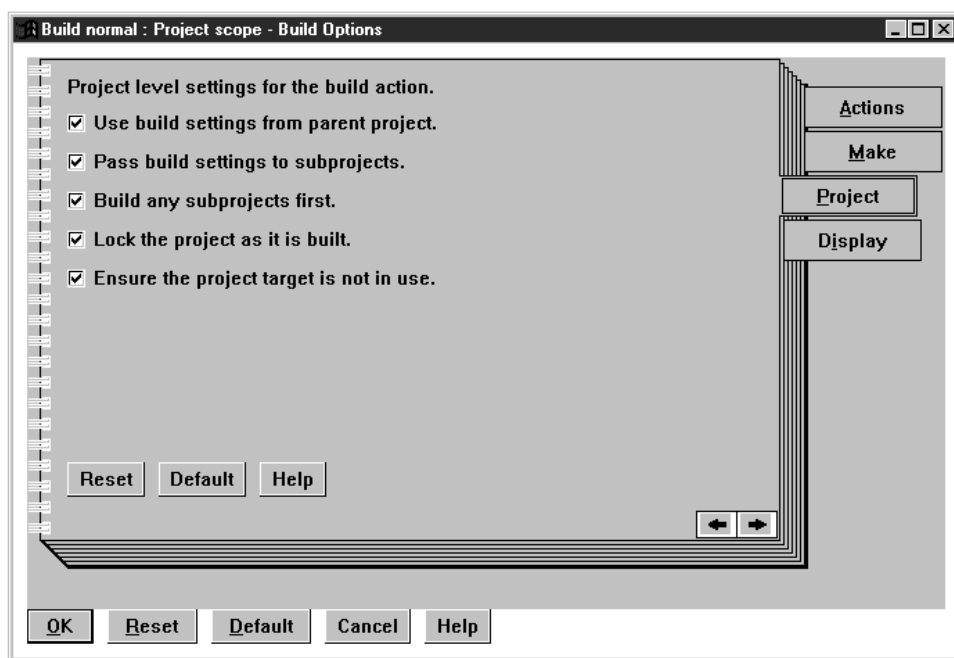



Figure 17. Build options - Project page

Most of the options on the **Project** page are only relevant if the project is part of a project hierarchy.  See "Organizing Projects" on page 19 for more information on how to use hierarchies to establish dependency relationships between projects.

## Building Your Target

### **Use Build settings from parent project**

Select this option to assume the Build settings from the parent project, if one exists. This feature enables a set of build actions to be specified at the root project level that are used by the root project and all its descendant projects, whether or not build actions have been configured for the subprojects. If the set of build actions for a descendant project needs to be different from those of its parent, explicitly set the build actions for the descendant project.


The build utility always uses the action options from the current project to build the target unless it has the **Use Build settings from parent project** setting selected.

### **Pass Build settings to subprojects**

Select this option to allow any nested projects to assume this project's Build settings. If you do not select this option, and the subprojects are set to use the Build settings from their parent, they will assume the Build settings from this project's parent instead, if one exists.

### **Build any subprojects first**

Select this option to have the Build utility build any subprojects before building this project's target. Select this option if your project has dependencies on the targets of the projects it nests.

Project hierarchies are built using a depth first search throughout the project tree. Any projects at the same level are built in an unspecified order. No dependencies should exist between projects on the same level in the project hierarchy.  See “Organizing Projects” on page 19 for more information on project hierarchies.

### **Lock the project as it is built**

Select this option to have the build utility ensure that another build will not run on the project at the same time. This ensures that the build will not fail because another process has gained access to the project's target while it is being built. If the target program is running when the Build utility is trying to build it, the build will fail because the target file is locked.

### **Ensure the project target is not in use**

Select this option to have the build utility check if the target can be accessed before starting the build. If the target is in use, the build will not start.

## Building Your Target

### Build Options - Display Page

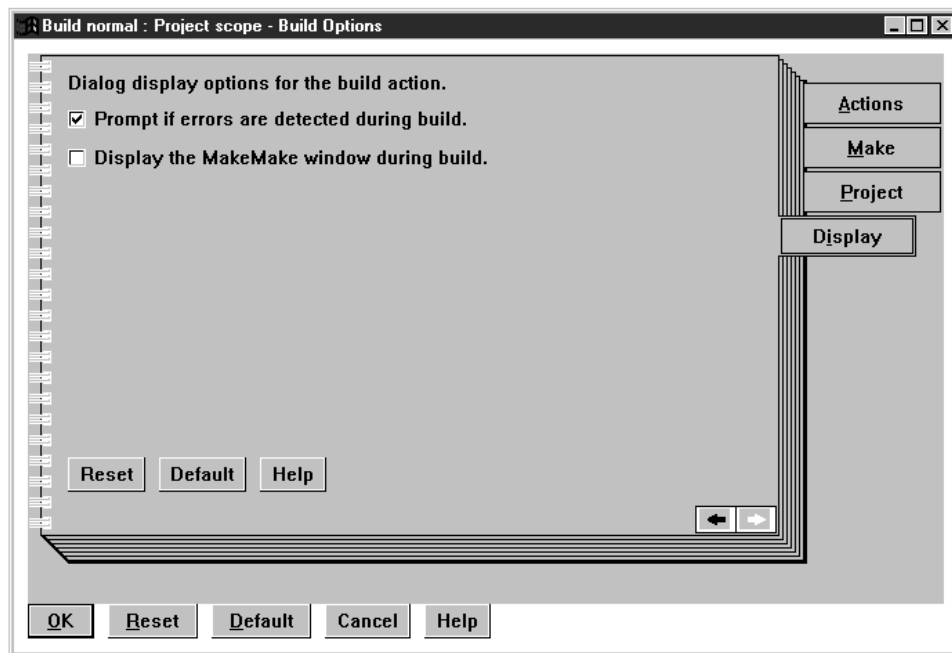


Figure 18. Build options - Display page

Use the display page to set options for prompts and windows to be displayed during the build.

## Building Your Target

### Prompt if errors are detected during build

Select this option to have the build utility display a message box when user action may be taken on an error. This is the default setting. If you do not select this option, any errors are written to standard out (viewable from the project monitor).

### Display MakeMake window during build

Select this option if you want the build utility to display the **MakeMake** window during the build so that you can generate the makefile explicitly before the build is run. The default is not to display the MakeMake window.

You will only need to select this option if all of the following are true:

- You selected the option to generate a makefile as part of the build (**Make** page).
- Your build actions were selected through MakeMake (that is, no build actions were selected from the **Actions** page).
- You think that the way the makefile is generated will change often between builds. For example, if you add another C++ file, you might want to add it to the source files selected in the **MakeMake** window. (See “Generating Makefiles” on page 52 for more information about the **MakeMake** window.) If you select your build actions from the **Actions** page, on the other hand, all applicable source files are processed automatically.

## Building Your Target

### Running Build from the Command Line

You can also run the Build utility from the command line. To invoke the Build utility from the command line, you need to specify the name of the project whose target to build, along with any other options you might need.

#### Command Line Notes:

- When you run the Build utility from the command line, specifying only the project name, the normal build action for the project is launched with the options that have been defined. You need to specify the optional command-line parameters only if you want to override the options already set for the action.
- Any options specified on the command line apply only to the root project of the build (not to any subprojects that may also be built). An exception is when the /PA (Pass build options to subprojects) option is specified for the root project, and the /U (Use build options from parent project) option is set in the subprojects.
- If you do not specify any build actions on the command line, and no build actions were set in the build action options, the Build utility uses the build actions from the last generated makefile. Failing that, it displays the **MakeMake** window, if the /D (Display) option is specified, so that you can select the build actions from there. If the /D option is not used, then the build fails since there are no build actions specified.

The general syntax of the Build utility is:

```
ibuild /PROJ[ECT] project [options]
```

#### Required Options:

*/PROJ[ECT] project*

The name of the project to build. You can name the project with its path name (for example, /PROJ D:\DESKTOP\MY\_PROJECT.IWP).

## Building Your Target

Makefile generation options:

`/T[OOLNAME] name`

The name of the project build action to be used in building the project's target.

`/A[CTION] action`

An action to be used in building the project's target. You can repeat this option as many times as there are build actions to specify. If no actions are specified, the actions and source files from the last successful makefile creation are used.

`/GENM[AKEFILE]`

Generate a makefile as part of the build. This is the default.

`/NOGENM[AKEFILE]`

Do not generate a makefile as part of the build; use the existing makefile.

`/GEND[EPFILE] [extension]`

Generate the dependency information in a separate file. You can also specify the dependency file filename extension. The generated dependency file will have the same name as the makefile, with a default extension of `.dep`.

`/NOGEND[EPFILE] [extension]`

Do not generate the dependency information in a separate file. The dependency information will be stored in the makefile.

`/GENS[CRIPT] script`

The makefile generation script to use. The default, `IWFMMGEN.CMD`, produces makefiles compatible with the `NMAKE` utility.

## Building Your Target

Make invocation options:

*/M[AKE] invocation command*

The make utility to run and its associated parameters.

*/GENF[ORCE]*

Create the makefile even if MakeMake detects that it has been user-modified.

*/NOGENF[ORCE]*

Prompt to determine whether or not to continue creating the make file if MakeMake detects that it has been user-modified. This is the default.

Project options:

*/B[UILDSUBPROJECTS]*

Build any descendant projects first. This is the default.

*/NOB[UILDSUBPROJECTS]*

Do not build any descendant projects.

*/U[SEPARENTOPTIONS]*

Use the build settings from the parent project, if one exists. This is the default.

*/NOU[SEPARENTOPTIONS]*

Do not use build settings from the parent project. Use this project's own build settings.

*/PA[SSOPTIONSTOSUBPROJECTS]*

Pass build settings to any subprojects.

*/NOPA[SSOPTIONSTOSUBPROJECTS]*

Do not pass build settings to subprojects. Subprojects should obtain their settings from this project's parent instead.



## Building Your Target

Miscellaneous options:

`/L[OCKPROJECTS]`

Lock the project while it is being built.

`/NOL[OCKPROJECTS]`

Do not lock the project while it is being built.

`/C[HECKTARGETS]`

Check if the target can be accessed before starting the build. If the target is locked, the build will not start.

`/NOC[HECKTARGETS]`

Do not check if the target can be accessed before starting the build.

`/P[ROMPT]`

Display a message box to query the user about an error. This is the default.

`/NOP[ROMPT]`

Do not display a message box to prompt on an error. Write the messages to standard out.

`/D[ISPLAY]`

Display the MakeMake window.

`/NOD[ISPLAY]`

Do not display the MakeMake window.

`/?`

Display help text for these options.

## Building Your Target

---

### The MakeMake Utility

Use the MakeMake utility to generate makefiles for your project. It can generate a makefile with the dependencies built in, or it can generate a separate dependencies file.

MakeMake creates a makefile for your project by examining the actions and source files associated with your project and then trying to determine the correct sequence of commands to build the project's target. Typically, in a hierarchy of projects, one makefile is generated per project. The Build utility handles the dependencies between projects and determines the order in which each project's make file should be processed to build the target of the current-level project.

### Generating Makefiles

The **MakeMake** window has two list boxes:

#### Actions

All the file-scoped actions in your project that are applicable building targets are listed here. Select the actions necessary for building the target of your project. For example, if your project builds a simple DLL, you might select the Compile and Link actions from the **Actions** list.

#### Source Files

All the project files or parts that match the source types for the selected actions are listed here. Select the source files that are to be processed by the makefile.

For example, if you have a project that builds a C++ executable with the C++ Compiler and Linker actions, select the .CPP source files.



## Building Your Target

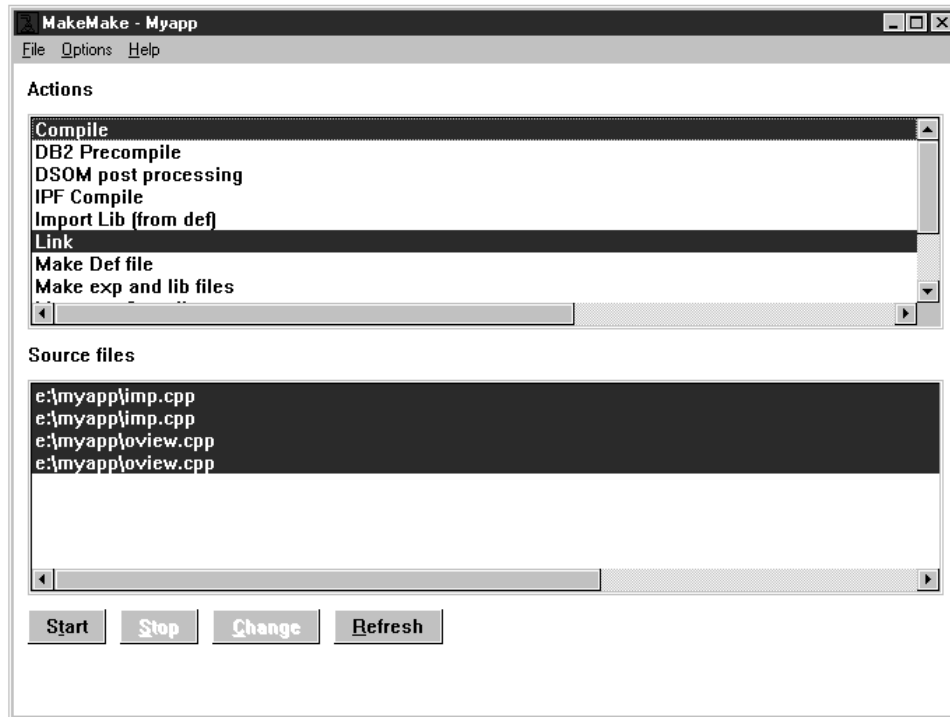


Figure 19. MakeMake window

When you have selected the actions and source files to build your target with, select the **Start** push button to start the makefile generation. Select the **Stop** button if you want to stop makefile generation. The **Refresh** button restores the selected actions and source files to the original set of actions and source files.

After a short while, the MakeMake **Results** window appears. It displays the generated makefile. The MakeMake **Results** window will not appear if the **Always show makefile** option on the MakeMake **Options** menu is not selected. You can display the **Results** window to view the generated makefile by selecting the **Change** push button from the MakeMake window.

## Building Your Target

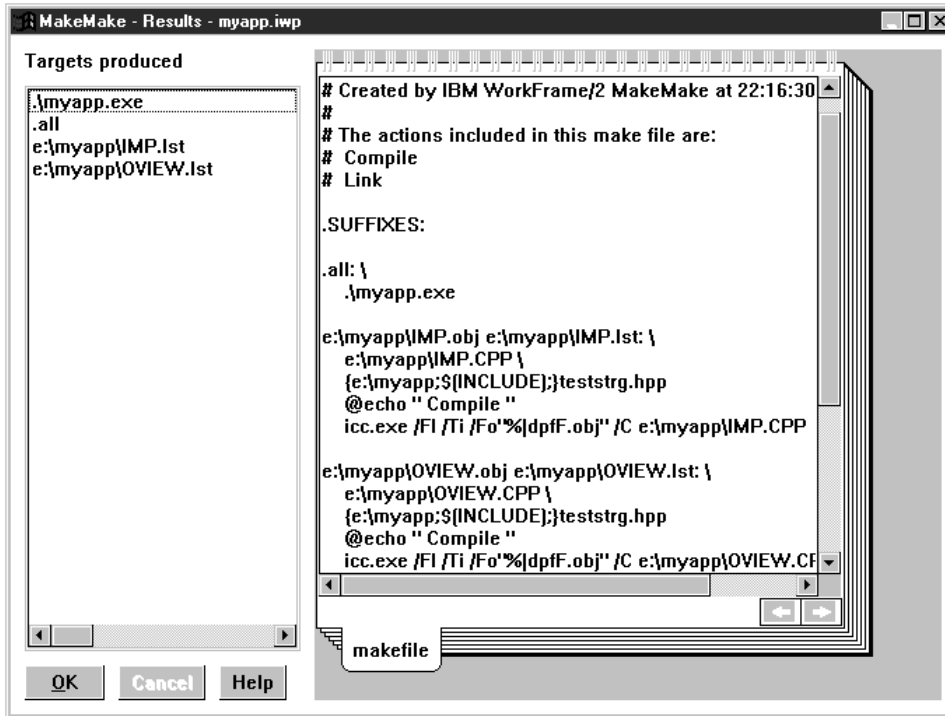


Figure 20. MakeMake Results window

The targets that are produced by each action in the make appear on a list box on the left side of the **Results** window. The makefile is displayed in notebook format on the right side of the window. You can edit the makefile from this window. If a separate dependency file was generated, use the notebook tabs to view the dependency file.

The **Cancel** button closes the results window, and returns you to the main MakeMake window without saving any updates to the makefile that you made. To save the makefile and close the makefile **Results** window, click on the **OK** button. This returns you to the main MakeMake window. Press **F4** to save the makefile and exit MakeMake. The **Help** button provides help for the MakeMake **Results** window.



MakeMake generates makefiles using build rules determined by examining the selected actions and source files. It does not use information from any existing makefiles in your project to generate a new makefile. If you have a makefile that you want to keep, rename it so that it is not overwritten by the newly generated makefile.

## Building Your Target

### Using MakeMake from the Command Line

The MakeMake utility can be invoked from the command line as well as from the project pop-up menu. To invoke MakeMake from the command line, optionally specify the name of the project to process, along with any other options you might want. If you do not specify a project name on the command line, the **MakeMake** window appears empty. You can load a project from the **MakeMake** window by selecting **Open project** from the **File** menu.

The general syntax of the MakeMake utility is:

```
imkmk [options]
```

The options are:

`/PROJ[ECT] project` The name of the project to be processed. You name the project with its path name (for example, `/PROJ D:\DESKTOP\MY_PROJECT.IWP`).

Makefile generation options:

`/A[CTION] action`

An action to be used in building the project's target. You can repeat this option as many times as there are make actions to specify. If no actions are specified, the actions from the project's default Build action are used. There are no Build actions specified, the actions and source files from the last successful makefile creation are used.

`/GENM[AKEFILE]`

Generate a makefile as part of the build. This is the default.

`/NOGENM[AKEFILE]`

Do not generate a makefile as part of the build; use the existing makefile.

`/GEND[EPFILE] [extension]`

Generate the dependency information in a separate file. You can also specify the dependency file's file name extension. The generated dependency file will have the same name as the makefile, with a default extension of `.DEP`.

`/NOGEND[EPFILE] [extension]`

Do not generate the dependency information in a separate file.

`/GENS[CRIPT] script`

The makefile generation script to use. The default, `IWFMMGEN.CMD`, produces makefiles compatible with the `NMAKE` utility.

`/GENF[ORCE]`

Create the makefile even if MakeMake detects that it has been user-modified.

`/NOGENF[ORCE]`

Prompt to determine whether or not to continue creating the make file if MakeMake detects that it has been user-modified. This is the default.

## Building Your Target

Miscellaneous options:

`/P[ROMPT]`

Display a message box to query the user about an error. This is the default.

`/NOP[ROMPT]`

Do not display a message box to report an error. Send messages to standard out. This option is only valid if `/NODISPLAY` is also specified.

`/D[ISPLAY]`

Display the **MakeMake** window. This is the default.

`/NOD[ISPLAY]`

Do not display the **MakeMake** window (execute the make file generation in batch mode). `/NOPROMPT` implies `/NODISPLAY`.

---

## Part 2. Compiling Your Program

This part of the *User's Guide* describes the input to the compiler, how to compile and link programs, how to set compiler options, and how to use the compiler listing. It also describes static and dynamic linking of programs.

---

<b>Chapter 5. Starting the Compiler</b> . . . . .	59
Compiling within WorkFrame . . . . .	59
Compiling from the Command Line . . . . .	60
Compiling from a Makefile . . . . .	62
 <b>Chapter 6. Controlling Compiler Input</b> . . . . .	63
Windows Environment Variables for Compiling . . . . .	66
Controlling <b>#include</b> Search Paths . . . . .	68
Setting the Source Code Language Level . . . . .	71
 <b>Chapter 7. Controlling Compiler Output</b> . . . . .	75
Using the Intermediate Code Linker . . . . .	81
Inlining User Code . . . . .	85
Setting the Calling Convention . . . . .	90
Choosing Your Runtime Libraries . . . . .	91
Using Precompiled Headers . . . . .	94
Controlling the Logo Display on Compiler Invocation . . . . .	102
Controlling Stack Allocation and Stack Probes . . . . .	102
 <b>Chapter 8. Setting Compiler Options</b> . . . . .	107
Specifying Compiler Options . . . . .	107
Using Parameters with Compiler Options . . . . .	109
Scope of Compiler Options . . . . .	111
Compiler Option Classification . . . . .	116
Compiler Options Summary . . . . .	117
Output File Management Options . . . . .	121
<b>#include</b> File Search Options . . . . .	128
Listing File Options . . . . .	130
Debugging and Diagnostic Information Options . . . . .	135
Source Code Options . . . . .	143
Preprocessor Options . . . . .	151
Code Generation Options . . . . .	154
System Object Model (SOM) Options . . . . .	168
Other Options . . . . .	171
Options for New ANSI Standards . . . . .	176

---

## Compiling Your Program





## Chapter 5. Starting the Compiler


The `icc` command invokes the VisualAge for C++ compiler, which takes your C or C++ source code as input and produces an intermediate code file, a preprocessed file, or an object file. The command also invokes the VisualAge for C++ linker to link the object file into an executable module or a dynamic link library (DLL). `icc` also invokes the ILIB utility to process your module definition files if you are outputting a DLL and do not have an export definition file.


You can invoke the compiler from:

- WorkFrame
- A makefile
- A Windows command line

You can also invoke it from within a program by using the `system` function. For example:

```
system("icc myprog.c");
```

 See the *C Library Reference* for information about the `system` function.

To compile without linking, use the `icc` command with the `/C+` option. Then you can link your program yourself, using the VisualAge for C++ linker.  See Part 3, “Linking Your Program” on page 177 for more information on linking.

---

### Compiling within WorkFrame

To use the compiler through WorkFrame, do the following:

1. Double click on the WorkFrame icon to open WorkFrame.
2. On the initial WorkFrame dialog, either open an existing project, or create a new project. These actions are also choices on the WorkFrame **Project** pull-down menu. Once a project has been opened or created, its files will be listed in the WorkFrame window.
3. Customize the compiler options from the **Options** pull-down menu, if the defaults are unacceptable. The **Options** menu contains choices that allow you to specify options for link and other actions. Use **Build Smarts** to easily set build options, such as debug, that affect more than one action. If you have not already done so, you may also want to customize the project settings by selecting **Settings** on the **View** pulldown menu.

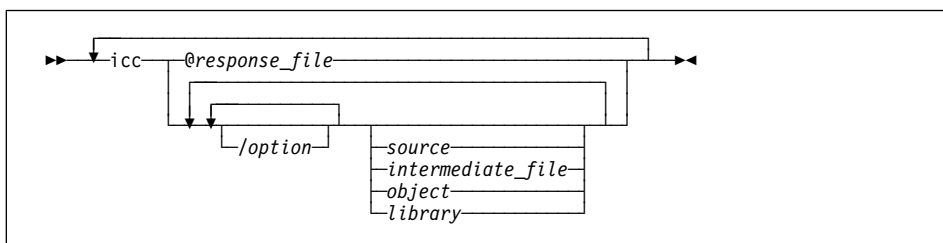
## Compiling from the Command Line

4. Select **Build** from either the **Project** pull-down menu or the project toolbar. Your project is built using the compiler as required.

---

## Compiling from the Command Line

The syntax for the `icc` command is as follows:




Depending on how you want to compile your files and how you have set up the ICC environment variable, many of the parameters used with the `icc` command are optional when you issue the command from a command line.

For example, to compile and link the program `bob.c`, you would enter the following:

```
icc bob.c
```

An object code file, `bob.obj`, and an executable file, `bob.exe`, are created.

For a list of all the VisualAge for C++ compiler options  see “Compiler Options Summary” on page 117. This summary is also available in the online version of the *User's Guide*. You can jump directly to the summary (or any other topic) from the command line with the `iview` command, followed by the book name (`cppug.inf`) and the topic name. For example:

```
iview cppug compiler options
```

You can also access an options summary with the `/?` option. Go to a command line and type:

```
icc /?
```

This listing is printed to **stderr**, but you can use the Windows redirection symbols to redirect it to **stdout** or to a file.

**Note:** The listing generated by this command is not intended to be used as a programming interface.

## Compiling from the Command Line

### Using Response Files

Instead of specifying compiler options and source files on the command line, you can use a *response file*. A response file is a text file that contains a string of options and filenames to be passed to `icc`. The string does not specify `icc` itself. For example, a response file that contains the single line:

```
/Sa /F1 catherine.c
```

would give the following command line:

```
icc /Sa /F1 catherine.c
```

A response file can have any valid filename and extension. To use the response file, specify it on the `icc` command line preceded by the at character (`@`). For example:

```
icc @d:\response.fil
```

No space can appear between the at character and the filename. You can use multiple response files, and even call another response file from within a response file. You can mix response files with other input on the command line. Options specified on the command line (including options specified in response files) take precedence over options specified in the ICC environment variable.



The command string in a response file can span multiple lines. No continuation character is required. The string can also be longer than the limit imposed by the Windows command line. In some situations you may need to use a response file to accommodate a long command line, such as when you use the intermediate code linker or compile C++ code containing templates.

Because the compiler appends a space to the end of each line in the response file, be careful where you end a line. If you end a line in the middle of an option or filename, the compiler may not interpret the file as you intended. For example, given the following response file:

```
/Sa /F  
1 catherine.c
```


the compiler would construct the command line:

```
icc /Sa /F 1 catherine.c
```

The compiler would then generate an error that the `/F` option is not valid, and would try to compile and link the files `1.obj` and `catherine.c`.

---

## Compiling from a Makefile

Use a makefile to organize the sequence of actions (such as compiling and linking) required to build your project. You can then invoke all the actions in one step. The NMAKE utility can save you time by performing actions on only the files that have changed, and on the files that incorporate or depend on the changed files.  See Chapter 63, “Program Maintenance Utility (NMAKE)” on page 709 for more information.

You can also use the /qmakedep compiler option to create a dependencies file that can be used with the NMAKE utility.

You can write the makefile yourself, or you can use WorkFrame to manage the makefile. When you build through WorkFrame, a makefile is created and maintained automatically.



## Chapter 6. Controlling Compiler Input

This section describes the methods you can use to control input to the compiler.

### Compiling Programs with Multiple Source Files

To compile programs that use more than one source file, specify all the filenames on the command line. For example, to compile a program with three source files (mainprog.c, subs1.c, and subs2.c), type:

```
icc mainprog.c subs1.c subs2.c
```

The source file containing the main module can be anywhere in the list. The executable output file will have the same name as the first source file but with the extension .exe. In the example above, the executable file will be mainprog.exe.

You can compile a combination of C and C++ files. For example:

```
icc cprog.c cppprog.cpp cxxprog.cxx othprog.oth
```



The file extension determines whether the file is compiled as a C file (.c or any other unrecognized extension) or as a C++ file (.cpp or .cxx). In the example above, cprog.c and othprog.oth are compiled as C files, and cppprog.cpp and cxxprog.cxx are compiled as C++ files.

You can also use the /Tc, /Tp, and /Td options to specify whether a file is a C or C++ file, regardless of its extension. The /Tc and /Tp options apply only to the filename immediately following the option, and specify that the file is a C file (/Tc) or a C++ file (/Tp). For example, given the following command line:

```
icc /Tc cprog.cpp cppprog.cpp /Tp cxxprog.c
```

cprog.cpp is compiled as a C file, and cppprog.cpp and cxxprog.c are compiled as C++ files.

The /Td option applies to all files that follow it on the command line. Use /Tdc to specify that all source and unrecognized files that follow are to be treated as C files, or /Tdp to specify that they are to be treated as C++ files. (You can specify /Td to return to the default handling of files.)

## File Types

Option	Behavior
--------	----------

/Tc	Compile next file as a C file.
/Tp	Compile next file as a C++ file.
/Tdc	Compile all subsequent source files and unrecognized files as C files.
/Tdp	Compile all subsequent source files and unrecognized files as C++ files.
/Td	Compile *.c and unrecognized files as C files. Compile *.cpp and *.cxx as C++ files.

For example, given the following command line:

```
icc /Tdp cxxprog.c othprog.oth /Td newprog.new
```

cxxprog.c and othprog.oth are compiled as C++ files, and newprog.new is compiled as a C file because /Td reset the default handling of files (files with unrecognized extensions are treated as C files).

## File Types

The VisualAge for C++ compiler uses file extensions to distinguish between the different types of files it uses. The default file extensions are:

<b>.asm</b>	assembler listing file
<b>.c</b>	C source file
<b>.cpp</b>	C++ source file
<b>.cxx</b>	C++ source file
<b>.ctn</b>	temporary file
<b>.def</b>	definition file
<b>.dll</b>	dynamic link library
<b>.exe</b>	executable file
<b>.hh</b>	SOM file
<b>.h</b>	C header file
<b>.hpp</b>	C++ header file
<b>.i</b>	preprocessor output file
<b>.l</b>	temporary file
<b>.lst</b>	listing file
<b>.lib</b>	library file
<b>.m</b>	temporary file
<b>.map</b>	map file
<b>.obj</b>	object file
<b>.res</b>	resource file
<b>.pch</b>	precompiled header file
<b>.pdb</b>	browser file
<b>.w</b>	intermediate file
<b>.wh</b>	intermediate file
<b>.wi</b>	intermediate file
<b>.wit</b>	temporary file

**.wli**            temporary file  
**.ws**            intermediate file

For example, when you are using VisualAge for C++ defaults, the command:

```
icc module1.c module2.obj mylib.lib
```

compiles the source code file `module1.c` and produces the object file `module1.obj`. When the linker is invoked, the object files `module1.obj` (created during this invocation of the compiler) and `module2.obj` (created previously), and the library file `mylib.lib` are passed to the linker. The result is an executable file called `module1.exe`.

If no extension is provided for a file, it is treated as an object file (`.obj`). You can change this default to `.c` by specifying the `/Sd` option.

## Using Wildcards in Filenames



You can use wildcards (`*` or `?`) in the name for any input file. Use `*` to stand for zero or more unknown characters. Use `?` to stand for exactly one unknown character.

For example,

```
icc module?.c my*.lib
```

compiles all source code files that begin with `module` plus one additional character (such as `module1.c` and `module2.c`) and generates object files with names derived from the source files (for example, `module1.obj` and `module2.obj`). When the linker is invoked, the object files are linked with all libraries that begin with `my` (such as `mylib.lib` and `mythr.lib`).

You cannot use wildcards to specify output filenames. Either accept the default output filenames, or specify an output filename in full.


## Windows Environment Variables

---

### Windows Environment Variables for Compiling

The VisualAge for C++ compiler checks the Windows environment variables for path information and for the default values of compiler options.


Environment variables can be set from the command line. In Windows 95, they can also be set in the AUTOEXEC.BAT file. In Windows NT, they can also be set in the **System** window. To get to the **System** window, double-click on **Main** and then on **Control Panel**) by clicking on **Set** to add a new environment variable to the list of User Environment Variables.

The environment variables described in this section are called the **compiler** environment variables. A number of environment variables are also used at run time.  See the *Programming Guide* for more information on the runtime environment variables.

The following Windows environment variables affect the operation of the VisualAge for C++ compiler during compilation.

**DPATH** Lists the directories that the compiler searches for message and help files.

**HELP** Lists the directories that the compiler searches for help (.hlp) files.



**ICC** Sets compiler options and filenames.  See “Specifying Compiler Options” on page 107 for a detailed description of the ICC variable.

**ILINK** Sets options that the VisualAge for C++ linker uses when it links the object files that the compiler generates. At link time, the options in this variable are processed before the options on the command line. If the compiler invokes the linker, any linker options specified through the compiler or the ICC variable are passed to the command line. If some options conflict, the option that was processed last takes effect.

**Note:** You cannot specify filenames in the ILINK environment variable. If you invoke the linker through the compiler, you can specify filenames for the linker in ICC.

**INCLUDE** Lists directories that the compiler searches for header files.

**LIB** Lists directories that the linker searches for library (.lib) files. This should include the directory that contains the VisualAge for C++ libraries, and the directory that contains the Toolkit's kernel32.lib library.

**LOCPATH** Lists directories that the **setlocale( )** function uses to locate locale data.  See the *Programming Guide* for more information on creating locales and using the **setlocale( )** function.  See Chapter 54, “LOCALDEF Utility” on page 609 for information on using the LOCALDEF utility.



## Windows Environment Variables

- PATH** Lists the directory (or directories, separated by semicolons) to be searched for executable files (and DLLs) when the compiler is invoked. This variable should include the directories containing VisualAge for C++ executables, for example the VisualAge for C++ compiler (`icc.exe`) and linker (`ilink.exe`) executable files.
- TMP** Sets the directory where the VisualAge for C++ compiler places all its temporary work files. This directory might also be used by other applications that generate temporary files. If this variable is undefined, the compiler uses the current directory. If you installed the compiler on a LAN, temporary files are stored in your local directory. The work files created by the compiler are normally erased at the end of compilation; however, if an interruption occurs during compiling, these work files may still exist after the compilation ends. If you set the TMP variable, you eliminate the possibility of work files being scattered around your file system.

### Filename in ICC

In addition to compiler options, you can also put filenames into the ICC variable. For example, if you specify:

```
SET ICC=test.c check.c
```

the command

```
icc main.c
```

causes `test.c`, `check.c`, and `main.c` to be compiled and linked, in that order. You can also specify intermediate files (created with the `/Fw` option) in ICC. They are treated like source files.

If you specify library (`.lib`) or object (`.obj`) files in ICC, they are passed to the linker when the compiler invokes it.

## Controlling #include Search Paths • #include Filename Syntax

---

### Controlling #include Search Paths

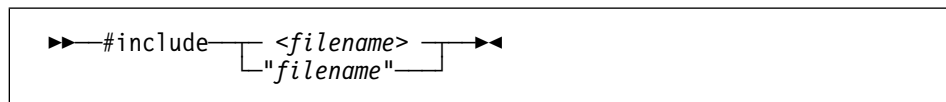
The **#include** preprocessor directive allows you to retrieve source statements from secondary input files and incorporate them into your program.

You can nest **#include** directives in an included file. You can have up to 127 levels of nesting in a C file (128 including the main level), and up to 255 levels of nesting in a C++ file (256 including the main level), when using the VisualAge for C++ compiler.

Compiler options and environment variables let you choose the disk directories searched by the compiler when it looks for **#include** files.

This section describes how to specify **#include** filenames and how to set up search paths for these files.

### #include Syntax



In the above figure, angle brackets indicate a **system #include** file, and quotation marks indicate a **user #include** file.

### #include File Name Syntax

You can specify any valid Windows filename in an **#include** directive. The filename must be sufficiently qualified (have enough of a path) for the compiler to be able to locate the file. In some cases, an unqualified or partially qualified filename may be sufficient, for example:

```
#include "..\HEADERS\myheader.h"
```

In other cases, you may have to include the entire path name.

If a path name is too long to fit on one line, you can place a backslash (\) as a continuation character at the end of the unfinished line to indicate that the current line continues onto the next line. The backslash can follow or precede a directory separator, or divide a name. For example, to include the following file as a user **#include** file:

```
c:\cset\include\mystuff\subdir1\subdir2\subdir3\myfile.h
```



you could insert one of the following **#include** directives in your program:

```
#include "c:\cset\include\mystuff\subdir1\sub\
dir2\subdir3\myfile.h"
```

or

```
#include "c:\cset\include\mystuff\subdir1\
subdir2\subdir3\myfile.h"
```


#### Notes:

1. The continuation character (\) must be the last non-white-space character on the line. (White space includes any of the space, tab, new-line, or form-feed characters.) The line cannot contain a comment.
2. The continuation character (\), although the same character as the directory separator, does not take the place of a directory separator or imply a new directory.

## Ways to Control the #include Search Paths

You can control the **#include** search paths in three ways:

- Use the `/I`, `/Xc`, and `/Xi` compiler options on the command line when invoking the compiler.
- Use the `/I`, `/Xc`, and `/Xi` compiler options in the ICC environment variable.
- Specify the search paths in the INCLUDE environment variable.

For more information on the compiler options `/I`, `/Xc`, and `/Xi`,  see “**#include** File Search Options” on page 128.

## #include Search Order

### #include Search Order

When the compiler encounters either a user or system **#include** file directive with a fully qualified file name (full path and filename), it looks only in the directory specified by the name.

#### User #include Files

When the compiler encounters a user **#include** file specification that is not fully qualified, it searches for the file in the following places, in order:

1. The directory of the original top-level file.
2. Any directories specified using /I that have not been removed with /Xc.  
Directories specified in the ICC environment variable are searched before those specified on the command line.
3. Any directories listed in the INCLUDE environment variable, unless you specified the /Xi option.

#### System #include Files

When the compiler encounters a system **#include** file specification that is not fully qualified, it searches for the file in the following places, in order:

1. Any directories specified using /I that have not been removed with /Xc.  
Directories specified in the ICC environment variable are searched before those specified in the command line.
2. Any directories listed in the INCLUDE environment variable, unless you specified the /Xi option.

## Accumulation of Options

The **#include** search options are cumulative between the ICC and INCLUDE environment variables and the command line. For example, given the following ICC and INCLUDE environment variables:



```
ICC=/I\rosanne  
INCLUDE=c:\kent;\alan
```

and the following command line:

```
icc /Xi+ /Ic:\connie test.c /Xi- /Xc /Id:\dal f:\moe\marko\jay.c
```

## Setting the Language Level

The **#include** files are processed as follows:

### User #include files

Any user **#include** files referenced in `test.c` are searched for first in the current directory, then in the directory `\rosanne`, and then in `c:\connie`. Because of the `/Xi` option, none of the directories in `INCLUDE` are searched.

Any user **#include** files referenced in `jay.c` are searched for in the following directories, in the given order: `f:\moe\marko`, `d:\dal`, `c:\kent`, and `\alan`. The directories in `INCLUDE` are searched because the `/Xi-` option overrides the `/Xi+` option specified previously. The `/Xc` option removes the directories `\rosanne` and `c:\connie` from the current search path.

### System #include files

Any system **#include** files referenced in the file `test.c` are searched for first in the directory `\rosanne` and then in the directory `c:\connie`. Because of the `/Xi+` option, none of the directories in `INCLUDE` are searched.

Any system **#include** files referenced in the file `f:\moe\marko\jay.c` will be searched for first in the `d:\dal` directory, then in the `c:\kent` directory, and finally the `\alan` directory. The directories in `INCLUDE` are searched because the `/Xi-` option overrides the `/Xi+` option specified previously. The `/Xc` option removes the directories `\rosanne` and `c:\connie` from the current search path.

---

## Setting the Source Code Language Level

You can set the language level of your source code to one of the following levels:

**ANSI** Compile according to a strict interpretation of the ANSI standard. Use this level when you want your code to be portable to other compilers. To achieve this conformity, you must also specify the `/qno long long`, `/Ss-`, and `Gf-` options.



**SAA Level 2** Compile according to the SAA Level 2 standard. Use this level when you want your code to be portable to other IBM compilers.



**Extended** Allow extended language features, allow non-standard language usage. Use this level when you are compiling code ported from another compiler, or when you are compiling code that does not need to be portable. If you will be compiling and running your code primarily on the workstation, you should use the extended language level.



**Compatible** Allow older versions of the C++ language. Use this level when you are compiling older code.

## Setting the Language Level

The levels are described in detail below. You can set the level using compiler options either on the command line or in the ICC environment variable, or by using a **#pragma langlvl** directive. Note that a **#pragma langlvl** directive in your source code overrides any conflicting compiler options. When you set the language level, you also define the macro associated with that level. The SAA C standards conform to the ANSI standards, but also feature some additional elements.

### ANSI

Allow only language constructs that conform to ANSI C standards or, for C++ code, that conform to the standards in the ANSI working paper on C++ standards. All non-ANSI constructs cause compiler errors. To achieve this conformity, you must also specify the `/qnoqlonglong`, `/Ss-`, and `Gf-` options.


Use this language level to write code that is portable across ANSI-conforming systems. If your code is error-free at this level, it should be error-free with any other compiler.

**Note:** Because VisualAge for C++ has a strict interpretation of the ANSI standard, it can find errors in code that compiles cleanly with other compilers. If you are porting code **into** VisualAge for C++, compile with the Extended language level.

To set this language level, use either the `/Sa` option or **#pragma langlvl(ansi)**, both of which define the macro `__ANSI__`.

### SAA Level 2



Allow only language constructs that conform to SAA Level 2 C standards. This language level is valid for C code only, because there is no SAA standard for C++. SAA constructs include all those allowed under the ANSI language level, because the SAA C standard conforms to the ANSI standard. All non-SAA constructs cause compiler errors. This language level supports some additional library functions, and specifies some behaviors that are left as implementation-defined by the ANSI standard.  See the *Language Reference* for a full description of the SAA C standard.


Use this language level to write code that is portable across SAA systems. If your code is error-free at this level, it should be error-free with any other IBM compiler.

To set this language level, use either the `/S2` option or **#pragma langlvl(saa12)**, both of which define the macro `__SAA_L2__`.

**Note:** You can also use **#pragma langlvl(saa)**, which defines the macro `__SAA__`. This level allows constructs that conform to the most recent SAA C definition. Because Level 2 is currently the most recent definition, the `__SAA__` and `__SAA_L2__` macros are equivalent at this time.

## Setting the Language Level

### Extended

Allow all VisualAge for C++ language constructs. These include all constructs that fall under the ANSI and SAA Level 2 language levels and the VisualAge for C++ extensions to those standards. This level also allows certain Windows library functions.  See the *Programming Guide* for more information.

Use the extended language level when you are creating code that does not need to be portable, or when you are porting code into VisualAge for C++ from another compiler or platform. Extended is the default language level.

To explicitly state this default (for example, on the command line to override a setting in ICC), use the `/Se` option or `#pragma langlvl(extended)`, both of which define the macro `__EXTENDED__`.

### Compatible



Allow constructs and expressions that were allowed by earlier levels of the C++ language. This language level is valid for C++ code only.

When the language level is set to compatible:

- Classes declared or defined within classes or declared within argument lists are given the scope of the closest non-class.
- typedefs and enumerated types declared within a class are given the scope of the closest non-class.
- The **overload** keyword is recognized and ignored.
- An expression showing the dimension in a delete expression is parsed and ignored. For example, given:  

```
delete [20] p;
```

  
20 is ignored.
- Conversions from `const void*` and `volatile void*` to `void*` are allowed. At other language levels, these conversions would require an explicit cast.
- Where a conversion to a reference type uses a compiler temporary type, the reference need not be to a `const` type.
- You can bypass initializations as long as they are not constructor initializations.
- You can return a void expression from a function that returns `void`.
- `operator++` and `operator--` without the second zero argument are matched with both prefix and postfix `++` and `--`.
- You can use the `$` character in identifiers. Note that you can also use `$` in C++ files when the language level is set to extended.
- In a cast expression, the type to which you are casting can include a storage class specifier, function-type specifier (`inline` or `virtual`), template specifier, or typedef. At other language levels, the type must be a data type, class, or enumerated type.

## Setting the Language Level

- You can have a trailing comma in a list of enumerators, for example, `enum E {e, };`.
- Given the expression `class A *a = new(x) A[100];`, the compiler looks for a member operator `new` because the placement syntax (`new(x)`) is used. The member operators are not typically used to allocate arrays.
- You can use the comma operator in a constant expression. This allows comma expressions to be used in places like case labels and array bounds, where they are normally prohibited.
- You can declare a member function using both the **inline** and **static** keywords, for example, `inline static void sandra :: pete(void);`. The **static** keyword is ignored.
- No error is generated if a function declared to return a non-void type does not contain at least one return statement. Such a function can also contain return statements with no value without generating an error.
- If two pointers to functions differ only in their linkage types, they are considered to be compatible types.

Use this language level to write code that is portable to systems with older implementations of C++, or to port older code to the VisualAge for C++ product.


To allow older C++ constructs, use the `/Sc` option or `#pragma langlvl(compat)`, both of which define the macro `__COMPAT__`.





## Chapter 7. Controlling Compiler Output


The VisualAge for C++ compiler can generate the following output:

- An object module for each C/C++ source file input.
- One executable module (or dynamic link library).
- An export definition file (or import library).
- A listing file for each C/C++ source file that contains information about the compilation.
- Preprocessed header files.
- Template-include files.  See the chapter on generating template-include files in the *Programming Guide* for more information about these files.
- A linker map file.
- A preprocessor output file for each C/C++ source file. You can use this output file for debugging information.
- An assembler listing file for each C/C++ source file. The format of the listing is in the style of the MASM 5.1 assembler input. The C/C++ source is annotated in the listing. Assembler listings will not always compile, especially if reserved MASM keywords are used as external variables or functions.
- A browser listing file for use by the VisualAge for C++ Browser.
- Intermediate code files. Four files (.w, .wh, .wi, .ws) are produced per source file.
- Dependency files that can be included in a makefile.
- Temporary files.
- Diagnostic information about possible programming errors.
- Messages (for example, the IBM logo and help messages).
- A return code (0 for a compile without errors).

**Note:** Any preprocessor output files, assembler listing files, intermediate code files, temporary files, or diagnostic information are not intended to be used as a programming interface.

## Compiler Output

### Object Files

The object files that are produced by VisualAge for C++ can be linked to create either executable (.exe) files or dynamic link libraries (.dll files). Use the /Ge+ option to create an executable file or /Ge- to create a DLL.  See “Code Generation Options” on page 154 for more information on using compiler options to specify the type of object file to be created.

**Optimizing Object Code:** The VisualAge for C++ compiler can perform many optimizations, such as local and global optimizations, function inlining, and instruction scheduling on object code. Use the /O+ option to generate code that is optimized for speed. By default, optimization is turned off (/O-). When you specify /O, you can also specify:


/Oc Optimize code for size as well as speed.

/Os Invoke the instruction scheduler. Turned on by default when you specify /O.



By default, /O also sets /Oi to inline user functions qualified with the **`_Inline`** or **`inline`** keywords.



The compiler can perform more complete optimization when you specify /O1 to invoke the intermediate code linker.

Specify /G1 to perform additional optimization during the linking step by removing unreferenced functions.  See Chapter 10, “Optimized Linking” on page 185 for more information on linker optimizations.

Use the /qtune option to tune your code for faster performance on a specific type of processor. By default, the compiler produces code that is tuned for all x86 processors.

 See “Code Generation Options” on page 154 for more information on using compiler options to control optimization. For more information on how you can optimize your code,  see the chapter on optimizing code in the *Programming Guide*.

## Compiler Output


**Generating Debugger Information:** The information necessary for running the VisualAge for C++ Debugger can be placed in the object file produced by the compiler using the `/Ti+` option. To include the debugger information in the executable file or DLL, use the `/DE` linker option. If you use `icc` to invoke the linker and specify `/Ti+`, the `/DE` option is automatically passed to the linker.




When you use `/Ti+`, do **not** turn on optimization ( `/O+`, `/Oc+`, `/Oi+`, or `/Os+`) unless you are using the information with the Performance Analyzer, and not with the Debugger. Because the compiler produces debugging information as if the code were not optimized, the information may not accurately describe an optimized program being debugged, which makes debugging difficult. Accurate symbol and type information is not always available.

If you cannot avoid debugging an optimized program, turn the scheduler off (`/Os-`), and step through the program at the assembly level, using the **Register** and **Storage** windows for information.

To make full use of the VisualAge for C++ Debugger, set optimization off (the default).

 See “Debugging and Diagnostic Information Options” on page 135 for more information on using compiler options to control the generation of debugging information.

 See Part 5, “Debugging Your Program” on page 261 for more information on the VisualAge for C++ debugger.

**Generating Performance Analyzer Information:** To include the information required by the Performance Analyzer in the object file, use both the `/Ti+` and `/Gh+` options. To include the Performance Analyzer information in the executable file or DLL, use the `/DE` linker option. If you use `icc` to invoke the linker and specify `/Ti+`, the `/DE` option is automatically passed to the linker.

## Compiler Output


When you specify /Gh+, the compiler generates a call to the profiling hook function (`_ProfileHook32`) as the first instruction in the prolog of each 32-bit function.

Other profiler vendors who plan to support the VisualAge for C++ product must provide their own profiling hook functions to gather all necessary runtime information.

**Generating Browser Information:** To create browser information, use the /Fb+ option to produce .PDB files that the Browser can use to display information about your program.

If you use `icc` to invoke the linker and specify /Fb, the /BROWSE option is automatically passed to the linker.

If you are compiling only, you must specify the /BROWSE option directly when you link.

 See “Creating Files to Use with the Browser” on page 335 for more information on generating browser files, and the differences between /Fb and /Fb\*.

## Executable Files

By default, the compiler generates one executable file for each compiler invocation. If you specify /C+, the compiler generates only object files, which you can then link separately to create an executable file.

There are two types of executable files:

- Those that run in the VisualAge for C++ runtime environment.

This is the default, and most C and C++ applications run under this environment. It supports all the VisualAge for C++ runtime functions and automatically provides initialization, exception management, and termination routines for C and C++.

- Those that run as subsystems.

Programs developed as subsystems can only make use of a subset of the VisualAge for C++ runtime library. You have to take care of initialization, exception management, and termination using Windows services and APIs.

Subsystems are intended for developing applications that cannot have a resident environment. If your application does not require the VisualAge for C++ runtime environment, you can also use the subsystem library to reduce your program's size and improve its performance. To compile a subsystem executable file, use the /Rn option.

## Compiler Output



For more information on subsystems and their uses, see the chapter on developing subsystems in the *Programming Guide*. For information on the compiler options used to produce subsystems, see “Code Generation Options” on page 154.

You can use several compiler options to change the executable file created by the compiler (see “Code Generation Options” on page 154 for more information).


### Compiler Listings

When you compile a program, you can produce a listing file that contains information about the source program and the compilation. You can use this listing to help you debug your programs.


**Note:** The compiler listing file is not intended to be used as a programming interface.

At the very minimum, the listing will show the options used by the compiler, any error messages, and a standard header that shows:

- The product number
- The compiler version and release number
- The date and time compilation commenced
- A list of the compiler options in effect

For information on how to use compiler options to specify the information and format of this file,  see “Listing File Options” on page 130.

### Temporary Files

The VisualAge for C++ compiler creates and uses temporary files during compilation. Temporary files are usually erased at the end of a successful compilation; however, if the compilation is interrupted, these files may be left on the disk. They are located in the path specified by the TMP environment variable. If you use memory files and they overflow to the disk, they will also be located in the path specified by TMP. If this variable is undefined, the compiler uses the current directory. For more information on the TMP variable,  see “Windows Environment Variables for Compiling” on page 66 and the chapter on run-time environment variables in the *Programming Guide*.

## Compiler Output

### Messages

You can use compiler options to control:

- The level of error message that the compiler outputs and that increments the error count maintained by the compiler (with the `/Wn` option).
- The number of errors that are allowed before the compiler stops compiling (with the `/Nn` option).
- The diagnostics run against the code (with the `/Wgrp` option).



See the online *User's Guide* for a list of compiler error messages.

See “Debugging and Diagnostic Information Options” on page 135 for more information on using the compiler options to control messages.

### Return Codes

The VisualAge for C++ compiler returns the highest return code it receives from executing the various phases of compilation. These codes are:

Code	Meaning
0	The compilation was completed, and no errors were detected. Any warnings have been written to <b>stdout</b> . Your executable file should run successfully.
12	Error detected; compilation may have been completed; successful execution impossible.
16	Severe error detected; compilation terminated abnormally; successful execution impossible.
20	Unrecoverable error detected; compilation terminated abnormally and abruptly; successful execution impossible.

If the error code is greater than 20, contact your IBM service representative.


For every compilation, the compiler generates a return code that indicates to the operating system the degree of success or failure it achieved.

## Precompiled Header Files • Using the Intermediate Code Linker

### Precompiled Header Files

You can use the `/Fi+` compiler option to create or recreate precompiled versions of header files used during that compilation.

To use the precompiled header files, specify the `/Si+` option. You can specify a name for the precompiled header object and a directory. If you do not specify a name or directory, the precompiled header files are stored in the current working directory, with the name `csetc.pch` (for C files) or `csetcpp.pch` (for C++ files).

For more information on generating and using precompiled headers,  see “Using Precompiled Headers” on page 94.

When you use precompiled header files, the following restrictions apply:

- You cannot use the same precompiled header file for C and C++ programs.
- To create a precompiled header file, the compiler process must have write permission to the directories you specify, or to the current working directories if none are specified. To use a precompiled header, the compiler process must have read permission for that file.
- Precompiled header files do not appear in any listing files.
- If you specify `/P+` to run the preprocessor only, the `/Fi` and `/Si` options are ignored.


---

### Using the Intermediate Code Linker


The intermediate code linker combines the information in all `.w`, `.wh`, `.wi`, and `.ws` intermediate code files into one set of files, which is then used by the compiler to optimize the code and generate a single object file.



In addition to the optimizations performed by the intermediate code linker itself, using this linker exposes more of your program to the optimizer at a time. The optimizer can then generate more efficient code. Using the intermediate code linker can result in improved code optimization, especially where inlining is used, and in better program performance. Note that using the intermediate code linker on code being compiled into an `.exe` file results in better performance improvements than if the same code were being compiled into a `.dll` file.

The intermediate code linker also performs more thorough error checking than the compiler can perform on its own.  See “Error Checking” on page 84 for more information.

## Using the Intermediate Code Linker

To use the intermediate code linker, specify the `/O1+` option on the `icc` command line. For best results, use the `/Gu` option, as  described in “Using the `/Gu` Option” on page 84, and specify `/O+` to turn optimization on.

**Note:** Because optimization limits the generation of debugging information, use `/O-` if you want to debug your program. The `/O1` option does not affect debugging information.



Given the following command:

```
icc /O+ /O1+ vij.c thomas.c tim.c
```

the compiler:

1. Compiles each source file into a set of intermediate code files (`.w`, `.wh`, `.wi`, and `.ws` files).
2. Invokes the intermediate code linker to link the intermediate code files of all three source files.
3. Optimizes the code.
4. Creates **one** object module for all three files and names it after the first file specified on the command line (`vij.obj`). (You can change the name of the object file using the `/Fo` option.)
5. Invokes the linker to create an executable module (`vij.exe`). (You can change the name of the executable file using the `/Fe` option.) If you want to link your object files separately, use the `/C+` option on the `icc` command line. You can then invoke the linker as you would for any other object file.

## Intermediate Code Files

Instead of creating an object file directly, you can use the `/Fw+` option to create and save the intermediate code files to be linked by the intermediate code linker at a later time. When you use `/Fw+`, compilation stops when the intermediate files are created. For example:

```
icc /Fw+ brian.c jim.c
```

creates only the intermediate files for `brian.c` and `jim.c`. No object or executable modules are created.

The `/Fw` option also takes an optional filename parameter that lets you specify the filename for the intermediate files. For example:

```
icc /Fwtony jeff.c
```

names the resulting intermediate files for `jeff.c` to `tony.w`, `tony.wh`, `tony.wi`, and `tony.ws`. Note that there is no space allowed between `/Fw` and the filename parameter.



## Using the Intermediate Code Linker

You can specify existing intermediate files on the `icc` command line to run the intermediate code linker and complete the compilation. You need only specify the name of the `.w` file; the `.wh`, `.wi`, and `.ws` files are included automatically. No option is required. For example, the command:

```
icc brian.w jim.w
```

links all intermediate files for `brian.c` and `jim.c`, creates an object file, and invokes the linker to create an executable module.

**Note:** You cannot use compiler options related to source files with intermediate files because the source has already been partially compiled. For example, you cannot produce a listing file from intermediate files or set the language level for the program.

You can also combine intermediate and source files on the command line to run the intermediate code linker on all the files and complete the compilation. No option is required. For example:

```
icc brian.w jim.c
```

## Restrictions

### Consistent Options

When you use the intermediate code linker, some options must be consistent across all files. Without the intermediate code linker, these options can be inconsistent, allowing you to compile different object files with different options. Because the intermediate code linker creates only one object file, you can no longer have these different options in effect. The following options must be consistent across all source files, when you use the intermediate code linker:

/G3	/G4	/G5
/Ge	/Gf	/Gh
/Gi	/Gr	/Gs
/Gw	/Nd	/O
/Oc	/Oi	/Om
/Op	/Os	/qtune
/qwin32s	/Re	/Rn
/Ti	/Tn	/W

### System Requirements

If you use the intermediate code linker on a large application, you will require more system resources than if you were simply compiling. For example, compiling and intermediate linking a 40 000-line application requires a working set of approximately 25M. If your executable module or DLL contains more than 100 000 lines of code, using the intermediate code linker is not recommended.

## Using the Intermediate Code Linker

### Using the /Gu Option

One of the optimizations performed by the intermediate code linker is to discard any defined data or functions that are:

- Not referenced in the files included in the link
- Not defined as exports either by the **\_Export** keyword or by **#pragma export**. Export functions when you are creating a DLL and want its functions to be available to other DLLs or .exe files.

If you call functions in files not included in the intermediate link, such as library functions or Windows APIs, this optimization cannot be performed because the data and functions could possibly be used by one of these external functions. Because library functions and APIs rarely use data defined in user code, the result is often poorly optimized code.



To ensure that all unreferenced data and functions are discarded, use the /Gu+ option. This option tells the intermediate linker that any external functions that are referenced will not use anything defined in the files being linked. Use the **\_Export** keyword to mark any definitions that will be used in a separate compilation unit (by another DLL, or .exe file).

In addition, /Gu+ causes all external functions and data that are not exported to be defined as static, which can result in better optimization.


### Error Checking


Another benefit of using the intermediate code linker is enhanced error checking of all files included in the intermediate link step. The intermediate code linker can find errors that would otherwise generate linker errors or unexpected runtime behavior, such as:

- Redefinition of variables and functions
- Inconsistent declarations or definitions of the same function (including differences in return type, linkage, number of arguments, and argument properties)
- Type mismatches between different declarations or definitions of the same variable, with the exception of:
  - Differences in integer type of the same length (int and long)
  - Some mismatches within structures and unions
  - Mismatches between array declarations where one of the declarations is an external reference

The intermediate code linker also checks for inconsistent compiler options that could cause conflicts. If the intermediate code linker warns you of conflicting options, either change the options to make them consistent or, if it is necessary to use the

## Inlining User Code

options inconsistently, turn off the intermediate code linker (/O1).  See “Restrictions” on page 83 for a list of options that must be consistent across source files.

The intermediate code linker generates compiler errors EDC6004 through EDC6025.  See the online *User's Guide* for explanations of error messages.

---

### Inlining User Code

By default, the compiler inlines certain library functions, meaning that it replaces the function call with the actual code for the function at the point where the call was made. These library functions are called intrinsic or built-in functions.

You can also request that the compiler inline the code for your own functions. There are two ways to inline user code:

1. Use the **`_Inline`** keyword to specify which functions you want to have inlined. You must specify the /O1 option to turn inlining on.



The C++ language provides the function specifier `inline` that you can use in the same manner as **`_Inline`**. The **`_Inline`** keyword is not supported for use in C++ programs.

2. Use the /O1 option with a *value* parameter to automatically inline functions smaller than the value specified.

**Note:** Requesting that a function be inlined makes it a candidate for inlining but does not necessarily mean that the function will be inlined. In all cases, the compiler ultimately decides whether a function is inlined.

### Using Keywords



For C files, use the **`_Inline`** keyword to qualify either the prototype or definition of the functions you want to have inlined. For example:

```
_Inline int james(int a);
```

specifies that you want james to be considered for inlining.



In C++ files, use the `inline` function specifier in the same way as **`_Inline`**. For example:

```
inline int angelique(char c);
```

specifies that you want angelique to be considered for inlining.

## Inlining User Code

The **\_Inline** and **inline** keywords have the same meaning and syntax as the storage class **static**. When you turn inlining on, the keywords also cause the functions they qualify to be considered for inlining. In addition, C++ member functions that are defined in a class declaration are considered candidates for inlining by the compiler.

## Using the /Oi Option

The /Oi option controls whether user functions are inlined or invoked through a function call:

- /Oi-        Do not inline user code. This is the default.
- /Oi+        Inline functions qualified with the **\_Inline** or **inline** keyword. When optimization is turned on (/O+), /Oi+ becomes the default.
- /Oivalue    Inline functions qualified with the **\_Inline** or **inline** keyword, as well as other functions that are smaller than or the same size as *value* in abstract code units (ACUs) as measured by the compiler. This option is called auto-inlining. In general, choosing the functions you want inlined yields better results than auto-inlining.

The /Oi option only affects user code, and does not affect the inlining of intrinsic VisualAge for C++ library functions. To disable the inlining of library functions, parenthesize the function call. For example:

```
(strcpy)(str1, str2);
```

In this way, you can selectively disable inlining of VisualAge for C++ functions.

Some library functions are implemented as built-in functions, meaning there is no code in the library. You cannot parenthesize calls to these functions.

See the *C Library Reference* for a list of all the intrinsic and built-in library functions.

You cannot selectively disable inlining for user functions: you can request that a function be inlined, but you cannot turn inlining on and then request that a specific user function **not** be inlined.

If you use auto-inlining, *value* has a range between 0 and 65 535 ACUs (abstract code units). The number of ACUs that constitute a function is proportional to the size and complexity of the function. Because the compiler calculates ACUs based on internal algorithms, you can only estimate the number of ACUs for a given function. The following code samples provide some examples on which you can base your estimates.

## Inlining User Code



The following function is 33 ACUs:

```
int florence(char a, int b)
{
    if(a != 10)
        b++;
    else
        b += 10;
    return(a);
}
```

The next function is 51 ACUs:

```
int sanjay(long par1, long par2)
{
    while(par1)
    {
        if(par2)
            test3();
        par1--;
    }

    if(par1)
        testing();
    par1 += par2;
}
```

When you compile, the compiler generates a message for each function it inlines based on the *value* you specified. Messages are not generated for functions qualified with **`_Inline`** or **`inline`**, or for C++ functions defined in a class declaration. For most applications, the most effective value for auto-inlining is between 5 and 20.

**Note:** The value required to inline a specific function may be slightly larger when `/0+` is specified than when `/0-` is specified.

When you turn inlining on for C programs, static functions that are called only once and are relatively small (16 ACUs or less) are also inlined. For this type of function, there is always a greater benefit in inlining. You can use `/0value` with a very small value to display the names of these functions. They are not inlined for C++ programs.

## Inlining User Code

### Benefits of Inlining

Inlining user code eliminates the overhead of the function call and linkage, and also exposes the function's code to the optimizer, resulting in faster code performance.

Inlining produces the best results when:

- The overhead for the function is significant; for example, when functions are called within nested loops.
- The inlined function provides additional opportunities for optimization, such as when constant arguments are used.



For example, given the following function:

```
void glen(int a, int b)
{
    if (a == 10)
    {
        switch(b)
        {
            case 1: .
                :
            case 20: puts("b is 20");
                    break;
            case 30: .
                :
            default: .
                :
        }
    }
}
```

and assuming your program calls `glen` several times with constant arguments, for example, `glen(10, 20)`;, each call to `glen` causes the `if` and `switch` expressions to be evaluated. If `glen` is inlined, the compiler can then optimize the function. The evaluation of the `if` and `switch` statements can be done at compile time, and the function code can then be reduced to only the `puts` statement from case 20.



The best candidates for inlining are small functions that are called often. Use the Performance Analyzer or a profiler to determine which functions to inline to obtain the best results.

To improve performance further:

- Use constant arguments in inlined functions whenever possible. Functions with constant arguments provide more opportunities for optimization.
- If you have a function that is called many times from a few functions, but infrequently from others, create a copy of the function with a different name and inline it only in the functions that call it often.
- Turn optimization on.

### Drawbacks of Inlining

Inlining user code usually results in a larger executable module because the code for the function is included at each call site. Because of the extra optimizations that can be performed, the difference in size may be less than the size of the function multiplied by the number of calls.


Inlining can also result in slower program performance, especially if you use auto-inlining. Because auto-inlining looks only at the number of ACUs for a function, the functions that are inlined are not always the best candidates for inlining. As much as possible, use the **`_Inline`** or **`inline`** keyword to choose the functions to be inlined.

When you use inlining, you need more stack space. When a function is called, its local storage is allocated at the time of the call and freed when it returns to the calling function. If that same function is inlined, its storage is allocated when the function that calls it is entered, and is not freed until that calling function ends. Ensure that you have enough stack space for the local storage of the inlined functions, in order to avoid a stack overflow.

### Restrictions on Inlining

The following restrictions apply to inlining:



- You cannot inline functions that use a variable number of arguments.
- For C++, you cannot declare a function as **`inline`** after it has been called.
- To use **`_Inline`** or **`inline`**, the code for the function to be inlined must be in the same source file as the call to the function. To inline across source files, you must either:
  1. Place the function definition (qualified with **`_Inline`**) in a header file that is included by all source files where the function is to be inlined.
  2. Use the intermediate code linker (with the `/O1+` option) and auto-inlining. The intermediate code linker is  described in “Using the Intermediate Code Linker” on page 81.
- Turn off inlining (`/O1-`) if you plan to debug your executable module. Inlining can make debugging difficult; for example, if you set an entry breakpoint for a function call but the function is inlined, the breakpoint will not work.
- The Performance Analyzer treats an inlined function as part of the function in which it is inlined.

## Setting the Calling Convention

- A function is not inlined during an inline expansion of itself. For a function that is directly recursive, the call to the function from within itself is not inlined. For example, given three functions to be inlined, A, B, and C, where:

- A calls B
- B calls C
- C calls back to B

the following inlining takes place:

- The call to B from A is inlined.
- The call to C from B is inlined.
- The call to B from C is not inlined because it is made from within an inline expansion of B itself.

---

## Setting the Calling Convention


The VisualAge for C++ compiler supports the following calling conventions:

**`_Optlink`**  
**`_System`**  
**`__cdecl`**  
**`__stdcall`**

The default is **`_Optlink`**. You can change the default with the `/M` option:

### Option Calling Convention

`/Ms`     **`_System`** calling convention  
`/Mc`     **`__cdecl`** calling convention  
`/Mt`     **`__stdcall`** calling convention  
`/Mp`     **`_Optlink`** calling convention (the default)

 See “Code Generation Options” on page 154 for more information on these compiler options.

You can also set the calling convention for individual functions using linkage keywords.

For example, to declare `kathryn` as a function with the **`_System`** calling convention, you could use the following statement:

```
int _System kathryn(int i);
```

You can also use **`#pragma`** directives to set the calling convention for C programs, but this is obsolete and may not be supported in future versions of the compiler.

For example:



## Choosing Runtime Libraries

```
#pragma linkage(kathryn, system)
```

Note that, when using the **#pragma linkage** directive, you must declare the function separately. Using linkage keywords is generally quicker and easier than using **#pragma linkage** directives.

Both the keywords and the **#pragma linkage** directive take precedence over a conflicting compiler option. If you use both methods and specify different conventions for the same function, an error message is generated.



The linkage keywords and **#pragma linkage** directive are described in more detail in the *Language Reference*. For more information on the calling conventions and how they work, see the *Programming Guide*.

---

## Choosing Your Runtime Libraries

When you compile, the compiler defines default VisualAge for C++ runtime libraries for the linker to use. You can use compiler options to control the linking process by changing the type of runtime library you link to. If you do not specify any options, the compiler uses the library that produces single-thread executable modules that are statically linked. You can link to another library by specifying the appropriate options. You would link to another library to:

- Dynamically link your program (discussed in the following section)
- Create a multithread executable module (📖 See the *Programming Guide* for more detailed information.)
- Develop a subsystem (📖 See the *Programming Guide* for more detailed information.)
- Create a DLL for use with another executable module. (📖 See the *Programming Guide* for more detailed information.)
- Create an executable or subsystem that can run in the Win32s runtime environment. (📖 See the *Programming Guide* for more detailed information.)

## Choosing Runtime Libraries

The naming conventions used for the libraries are intended to help identify their function. The libraries are named as follows:

Figure 21. VisualAge for C++ Library Naming Conventions

Character Position				Significance
1 - 4	5	6-7	8	
CPPW				Product prefix
	S M N P Q			Single-thread library Multithread library Subsystem library (no runtime environment) Win32s library Win32s subsystem library
		35		Version of VisualAge for C++
			I O	Import library Object library (contains initialization routines) Statically bound library (no eighth letter)

For example, the library CPPWS35.LIB is the standard single-thread library for building both executable modules and DLLs, while CPPWN35I.LIB is the standard import library for creating a subsystem.

## Static and Dynamic Linking

**Static linking** means that code for all the VisualAge for C++ runtime functions called in your program is copied from a .lib file into your output .exe or .dll file. The .exe or .dll files will be larger because there is a copy of the runtime functions in each file. These programs will take up more storage, and if you run them at the same time, there will also be a copy of the library functions in memory for each program. Statically linked programs, however, are easier to distribute because the library functions are part of your executable file. See Note 1 below.

**Dynamic linking** means that code for the VisualAge for C++ runtime functions called in your program is **not** copied into your output .exe or .dll file. Instead, the function code stays in a separate VisualAge for C++ .dll file, and your calls to the function are resolved at load time. The amount of disk space required by your .exe or .dll file is reduced, and there is only one copy of the library functions in memory for all programs that use them. Dynamically linked programs can be harder to distribute, since the separate .dll file must be distributed along with your executable file.




Use the /Gd compiler option to control whether your executable file links to the runtime library statically or dynamically.



The default is /Gd-, which statically links with the .lib version of the runtime library.

## Choosing Runtime Libraries


Specify `/Gd+` to dynamically link with the DLL version of the runtime library.

The compiler option you choose causes the corresponding library to be linked in by default. If you override the default libraries with the `/NOD` linker option, you must explicitly give the name of all libraries you are using on the linker command line.

You can also statically or dynamically link to other libraries. For more information,  see Chapter 12, “Linking with Library Files” on page 193.


Under the VisualAge for C++ licensing agreement, you cannot ship the VisualAge for C++ DLLs as they are with a product that you develop. If you want to dynamically link to the VisualAge for C++ library, you can create your own version of the VisualAge for C++ runtime DLLs, as  described in the *Programming Guide*, or you can use the DLLRNAME utility,  described on page 543, to rename the VisualAge for C++ DLLs before you ship them.

### Notes:

1. You can also link dynamically to your own DLLs. Creating and using your own DLLs is  discussed in “Producing a Dynamic Link Library” on page 191, “Linking to Dynamic Link Libraries” on page 194, and in the *Programming Guide*.

## Using the Multithread Library

More than one thread may use the same runtime functions. To avoid contention for internal resources, the library ensures that only one thread at a time is active in the critical section of a function. Although this support is mandatory in a multithread program, it is unnecessary in a single-thread program.

This section describes only the compiler options you use to choose the single-thread or multithread version of the library. There is more information on creating a multithread program  in the *Programming Guide*.

If you want to create an executable file with multithread capabilities:

1. Specify the `/Gm+` option when you compile.
2. Use the multithread library when you link the object files.


## Using Precompiled Headers

If you want to create an executable file designed for a single thread only:


1. Use the default option `/Gm-` when you compile.
2. Use the single-thread library when you link the object files.

The compiler option you choose causes the corresponding library to be linked in by default. If you override the default libraries with the `/NOD` linker option, you must explicitly give the name of all libraries you are using on the linker command line.

## Enabling Subsystem Development

If you are creating a subsystem, specify the `/Rn` option to select the subsystem libraries.  See page 167 for a description of the option.

Functions in the subsystem libraries are intended for use in single-thread applications only. No multithread support is provided. If you want to use the subsystem libraries in multithread programs, you must provide your own protection and serialization using Windows semaphores. You must also provide your own buffering for input and output.

 See the *Programming Guide* for information on developing subsystems.

---


## Using Precompiled Headers

You can improve your compile time by using precompiled headers. Use the options `/Fi+` and `/Si+` together to automatically create and maintain precompiled header files for your application.

If you use the options consistently, precompiled header files are created if they do not exist, and used if they do. When a source file is changed, the precompiled version is automatically regenerated.



The compiler generates a single precompiled object for the first **initial sequence** of **#include** directives. The next time you compile, this single object can be used wherever that initial sequence appears. Since the precompiled object is only used in cases where the context is the same (same language, same beginning sequence of **#include** directives, compatible options and macro definitions), the precompiled object does not have to be reinterpreted every time it is included.

To get the most benefit from this new method, use the same initial sequence of headers wherever possible. The more files that share the same initial sequence, the greater the improvement in your compile time.  See “Organizing Your Source Files” on page 100 for tips on getting the most improvement.

You can specify different names or directories for precompiled header files with the `/Fi` and `/Si` options, or using **#pragma hdrfile**. This allows you to create more than

## Using Precompiled Headers

one initial sequence, and further improve your compile time. If you do not specify a name or directory, the precompiled headers are stored in the current working directory, with the name `csetc.pch` (for C files), or `csetcpp.pch` (for C++ files).

### Determining the Initial Sequence

The initial sequence of headers can consist of the following:

- **#include** directives
- comments
- **#error** directives
- null directives
- false conditional compilation blocks beginning with **#elif** or **#else**
- **#endif** directives

The first **#include** directive can be preceded only by comments and preprocessing directives. If it is preceded by anything else, then the compiler does not create or attempt to use precompiled headers with that source file.

The initial sequence is ended by any construct not in the above list. You can also stop the initial sequence with **#pragma hdrstop**. If you use **#pragma hdrstop** before the first **#include** directive, there is no initial sequence.



Any **#include** directives after the initial sequence are not precompiled: they will be compiled every time you compile the source file.

**Note:** When a header contains conditional compilation directives to prevent it from being included a second time, it is only counted once in the initial sequence, even if it appears multiple times.

## Using Precompiled Headers



### Example

```
main.c                                     h1.h
-----
/* Comments are OK */                     int h1;
#define M 1                               #include "h3.h"
#undef N
#line 10
#if F
    int f(int);
#endif
#if STDIO
    #include <stdio.h>
#endif
#include "h1.h"
/* Comments are OK */
#
#include "h2.h"
#include "h3.h"
main() {
}

h2.h
-----
int h2;

h3.h
-----
#ifndef H3_H
#define H3_H
    int h3;
#endif
```

The initial sequence can vary, depending on whether any macros are defined on the command line.

Macros defined	Resulting initial sequence
None	"h1.h", "h2.h", "h3.h"
STDIO	<stdio.h>, "h1.h", "h2.h", "h3.h"
F	No initial sequence (because <code>int f(int)</code> occurs before any <b>#include</b> directives)

Although `h3.h` is included twice (once in `main.c` and once in `h1.c`), only the first **#include** is considered in the initial sequence, because the second **#include** does not take effect.

## Matching the Initial Sequence

Once the precompiled initial sequence is created, it can be used by other compilation units (in subsequent compiles). Other compilation units can use the precompiled initial sequence under the following conditions:

- The compilation unit has a matching initial sequence of **#include** directives. The compilation unit can have a longer initial sequence, as long as the first part of the sequence matches. Any **#include** directives beyond the initial matching portion are compiled normally.
- The files that make up the precompiled header object have not changed. The compiler checks the modification date on each file.

## Using Precompiled Headers

- Any macros that were expanded or tested while generating the precompiled header object are defined with the same replacement tokens. The compiler checks macro names that are:
  - Defined before the start of the initial sequence, using **#define** or the **/D** option
  - Undefined before the start of the initial sequence, using **#undef** or the **/U** option
  - Predefined by the compiler

If the macro was not expanded or tested during the precompile, then the status of the macro does not matter, and does not have to match.

- No additional macros have been defined.
- The same compiler options are in effect.

### Example

Given two compilation units, prog1.c and prog2.c:



prog1.c	prog2.c
-----	-----
#undef X	#define X 1
#include "h1.h"	#include "h1.h"
#include "h2.h"	#include "h2.h"
func1() {}	func2() {}
h1.h	
-----	
#if TEST	
int h1;	
#endif	
h2.h	
-----	
char h2 = M;	

The file prog2.c can use the precompiled header object from prog1.c when:

- TEST has the same definition in both prog1.c and prog2.c, or is not defined in both.
- M has the same definition in both prog1.c and prog2.c, or is not defined in both.
- No additional macros have been defined in prog2.c (whether they are used or not).

The different definitions of X in prog1.c and prog2.c do not matter, since X is never tested or expanded.

## Using Precompiled Headers

### Using Multiple Initial Sequences



Because of the restrictions on reusing precompiled headers (same sequence of headers, same context in terms of macro names and options), you may want to use more than one precompiled header object.

You can specify the name of an alternative precompiled header file to use, or an alternative directory to search, with either of two methods:

- With the options `/Fi+` and `/Si+` on the command line. If the options specify different filenames, the compiler uses the last one specified for both options.
- With **#pragma hdrfile** in the source file, before the first **#include** directive. The pragma only takes effect if you specify at least one of `/Fi` or `/Si`.

If you specify a filename with both the options and the **#pragma**, the **#pragma** filename is used.

The default filenames are:

- `csetc.pch` for C compiles
- `csetcpp.pch` for C++ compiles

The default directory is the current working directory.

### Examples

The following examples show the interaction of options and **#pragma hdrfile** directives:



#### Example 1

```
#pragma hdrfile "fred.pch"
#include "h1.h"
#include "h2.h"
main () {}
```

#### Options Specified

`/Fi+ /Si+`

`/Fidave.pch`

#### Behavior

The headers "h1.h" and "h2.h" are precompiled using the file "fred.pch"

The compiler ignores the name specified with the option, because it is in conflict with the **#pragma hdrfile** directive. The compiler generates new headers in "fred.pch", but does not use them.



## Using Precompiled Headers



### Example 2

```
#pragma hdrfile "fred.pch"
#include "h1.h"
#pragma hdrstop
#include "h2.h"
main () {}
```

#### Options Specified

/Fi+ /Si+

/Sidave.pch

#### Behavior

Only the header "h1.h" is precompiled, using the file "fred.pch"

The compiler ignores the name specified with the option, because it is in conflict with the **#pragma hdrfile** directive. The compiler looks for the precompiled headers in "fred.pch", but does not generate new headers.



### Example 3

```
#include "h1.h"
#pragma hdrstop
#include "h2.h"
main () {}
```

#### Options Specified

/Fi+ /Si+

/Sidave.pch /Fijohn.pch

#### Behavior

Only the header "h1.h" is precompiled, using the file "csetc.pch" (for a C file) or "csetcpp.pch" (for a C++ file)

The compiler ignores the name specified with /Si, and uses the file "john.pch", which is specified later. The compiler looks for precompiled headers in "john.pch" and regenerates them if they are not found or are out of date.

## Using Precompiled Headers



### Example 4

```
#pragma hdrstop
#include "h1.h"
main () {}
```

#### Options Specified

Any options

#### Behavior

No headers are precompiled, because there is no initial sequence: **#pragma hdrstop** occurs before the first **#include** directive.

## Organizing Your Source Files



To take full advantage of the precompiled header improvements, you may need to reorganize your source files. Using precompiled headers without organizing your source files can actually **slow** your compilation.

There are several strategies you can use to organize your files:

### A precompiled header for each compilation unit

Use **#pragma hdrfile** in each primary source file to specify a distinct precompiled header object for each compilation unit.

#### Benefits

Each compilation unit has its own precompiled headers, so you can create the longest possible initial sequence for each compilation unit. You do not need to match the initial sequences in other compilation units, because you are not sharing the precompiled header object.

#### Drawbacks

If you change one header file, the compiler regenerates every precompiled header object that includes that header. This method can also require large amounts of disk space, since many precompiled headers are generated: a common header is precompiled separately for every compilation unit that includes it.

### Single header file

Create a single header file that has **#include** directives for every header in your application, and include it in each primary source file.

#### Benefits

You get the maximum possible benefit from using precompiled header files, and the maximum possible improvement in compile time. You have an exact match of the initial sequence for every compilation unit.

#### Drawbacks

## Using Precompiled Headers

If you are using a program maintenance utility such as NMAKE, you will end up recompiling the entire application every time you change even a single header file. For larger applications, this is probably unacceptable.

### Global header file

Create a single header file that has **#include** directives for those header files that are shared by many different compilation units. Include this global header file as the first step of the initial sequence in each primary source file, followed immediately by **#pragma hdrstop**.

#### Benefits

You get the benefit from precompiling common and shared headers, without having to recompile when you change less common headers.

#### Drawbacks

There is only one group of precompiled headers. Any headers outside of that group are compiled normally, and the precompiled header object must be regenerated every time you change even one of the common headers.



### Grouping headers

Do the following:

1. Identify headers that are common throughout your source, and divide them into smaller groups of associated headers. A header can be included in more than one group.
2. For each group, create a header that contains **#include** directives for each header in the group.
3. In each primary source file, identify the precompiled header filename with **#pragma hdrfile**, **#include** the appropriate group header, and end the initial sequence with **#pragma hdrstop**. If a source file does not use any of the precompiled header groups, put **#pragma hdrstop** as the first directive in the file.

#### Benefits

Common headers are precompiled. Changing a header only affects the groups it belongs to, rather than requiring all precompiled headers to be regenerated. This method does not use as much disk space as having a precompiled header object for each compilation unit, and is more maintainable than a single global header file.

#### Drawbacks

Requires additional work to organize headers into groups.

## Controlling Stack Allocation and Stack Probes

---

### Controlling the Logo Display on Compiler Invocation

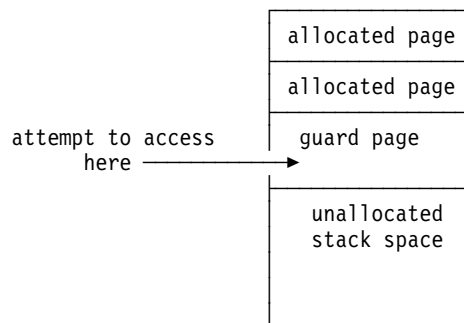
By default, the VisualAge for C++ logo appears on **stderr** when the compiler is invoked. You can stop the logo from appearing on `icc` invocation by specifying the `/Q+` option. To request explicitly that the logo appear, specify the `/Q-` option.

---

### Controlling Stack Allocation and Stack Probes

By default, the linker reserves 1 MB of stack space and commits (physically allocates) 1 page. The page with the largest address is committed, and the page below it is set up as a *guard page*. No other pages are committed.

When the guard page is accessed, an *out of stack* exception (STATUS\_GUARD\_PAGE\_VIOLATION) is generated. The system responds by attempting to get another guard page below the one previously allocated:<sup>1</sup>

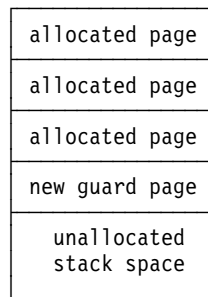


---

<sup>1</sup> For the purposes of this discussion, the stack grows down.

## Controlling Stack Allocation and Stack Probes

If this attempt is successful, the original guard page becomes a normal stack page and the next uncommitted page becomes the new guard page.



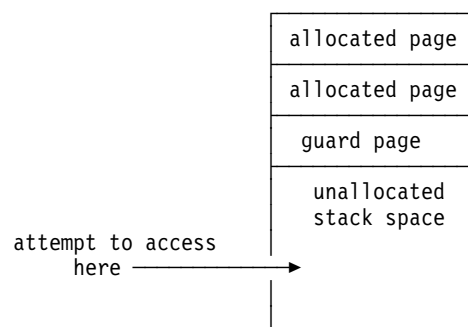
This process continues until a new guard page can no longer be allocated.

If the system cannot set a new guard page because it has reached the size limit of the stack (see “Setting the Stack Size” on page 105), a *guard page allocation failure* exception (STATUS\_STACK\_OVERFLOW) is generated. (The same exception is generated when the `_alloca` function runs out of memory.)

### Using Stack Probes

For the stack growth mechanism to work correctly, each 4K page must be accessed in the correct order. To ensure the correct access, the VisualAge for C++ compiler generates one or more stack probes in the prolog of each procedure that has automatic storage greater than 4K.

When a guard-page exception occurs, the stack probe instructions allow the exception mechanism to enlarge the stack if necessary. If an attempt is made to access the stack below the guard page:



stack probes cause the operating system to allocate each page of the stack up to that access point and to create a new guard page:

## Controlling Stack Allocation and Stack Probes

allocated page
allocated page
allocated page
allocated page
new guard page

Without stack probes, accessing the stack below the guard page is an access violation (you cannot access uncommitted pages). The process terminates. The compiler ensures that structures greater than 4K that are passed by value are placed on the stack to allow this mechanism to work.

Support for automatic stack growth is provided by default as needed.

**Note:** The `_alloca` function allocates storage on the stack. Unless you specify the `/Gs+` option, the compiler generates stack probes to allocate the required memory.

You do not need to use stack probes if:

- You have changed the linker default for the amount of committed memory and are sure that all local variables require less than this amount.
- You can guarantee that the stack will always be allocated. For example, you could write a guard routine to run once at the beginning of each thread and serially access each page up to the last page, leaving that page as a guard page.
- Your local variables require less than 4K of storage on the stack.

To turn off stack-probe generation, specify the `/Gs+` compiler option. (See page 158 for the option description.) Because stack probes go into the prolog of every function with more than 4K of stack storage, your program will run faster with the stack probes turned off. However, it is only safe to turn off stack probes if you can meet one or more of the above criteria.

## Setting Stack Size

### Setting the Stack Size

You can set the stack size in one of two ways:

1. Specify the `/B"/STACK:reserve[,commit]"` compiler option.
2. Specify the `/STACK: reserve[,commit]` linker option on the linker command line.

By default, the linker reserves 1 MB of stack space and commits (physically allocates) 1 page. Setting the stack size using one of the options listed above overrides the default value. For example, specifying the linker option

```
/STACK:65536
```

sets the stack size to be 64K.

## Setting Stack Size





## Chapter 8. Setting Compiler Options


You can use compiler options to alter the compilation and linking of your program. This chapter describes these options and tells you how to use them.

---

### Specifying Compiler Options

Compiler options are not case-sensitive, so you can specify them in lowercase, uppercase, or mixed case. You can also substitute a dash (-) for the slash (/) preceding the option. For example, -Rn is equivalent to /Rn. Lowercase and uppercase, dashes, and slashes can all be used on one command line, as in:

```
icc /ls -RN -gD /Li prog.c
```


Some options have parameters.  See “Using Parameters with Compiler Options” on page 109 for information.

You can specify compiler options in the following ways:

- On the command line
- In the ICC environment variable
- In the WorkFrame environment

Options specified on the command line override the options in the ICC variable.

### Setting Options on the Command Line

Compiler options specified on the command line override any previously specified in the ICC environment variable (as described below and  in “Windows Environment Variables for Compiling” on page 66).

For example, to compile a source file with the multithread option, enter:

```
icc /Gm myprog.c
```

### Setting Options in ICC

Frequently used command-line options can be stored in the ICC environment variable. This method is useful if you find yourself repeating the same command-line options every time you compile. You can also specify source filenames in ICC.

## Specifying Compiler Options

The ICC environment variable can be set from the command line. If it is set on the command line, the options will only be in effect for the current session.

For example, to specify that a source listing be generated for all compilations and that the macro `DEBUG` be defined to be 1, use the following command at the Windows prompt.


```
SET ICC=/Ls+ /DDEBUG::1
```

For any option that includes an equals sign in the syntax, use a double colon (::) instead of the equals sign, because the equals sign is not allowed in Windows environment variables.

Now, type `icc prog1.C` to compile `prog1.C`. The macro `DEBUG` is defined as 1, and a source listing is produced.

Options you specify on the command line override options in the ICC variable. For example, the following compiler invocation voids the effect of the ICC setting in the last example:

```
icc /Ls- /UDEBUG fred.c
```

 See “Windows Environment Variables for Compiling” on page 66 for more information about using ICC and other environment variables.

## Setting Options in the WorkFrame Environment

If you have installed the WorkFrame product, you can set options as follows:

1. Open the **Project** window by double-clicking on the WorkFrame icon and opening a project file.
2. Select **Build Smarts** from the **Options** pulldown menu. The **Build Smarts** window appears, in which you can add support for VisualAge for C++ features like the Debugger. **Build Smarts** modifies the options for all of the VisualAge for C++ components affected by the chosen feature. For example, when the Debugger is selected in the **Build Smarts** window, **Build Smarts** adds the Debugger options to both the compiler and linker.

If you want to set options on an individual basis, do the following:

- a. Select **Compiler** from the **Options** menu to display the **Compiler Options** notebook.
- b. Select options in the notebook. Turn the pages to see all the options (there can be several pages under one tab).


If you prefer to set options in the command-line form, turn to the **User** tab of the notebook, and use the entry field there to enter options as you would on the command line.

## Compiler Option Parameters

See the online version of the *User's Guide*, and the online help for the notebook, for a mapping of command-line options to WorkFrame options.

- c. Select **OK** when you are done.

The next time you build your project, the options you selected are used.

For more information on compiling with WorkFrame,  see “Compiling within WorkFrame” on page 59.

---

## Using Parameters with Compiler Options

For all compiler options that take parameters, the following rules apply:

- If a parameter is required, you can put zero or more spaces between the option and the parameter.  
For example, both `/FeMyexe.exe` and `/Fe Myexe.exe` are valid.
- If a parameter is optional, do not put spaces between the option and parameter.  
For example, `/FlMylist.lst` is valid, but `/Fl Mylist.lst` is not.
- For the `/q...` options that take more than one suboption or parameter, the suboptions or parameters must be separated by colons. The uppercase letters in some suboptions indicate the minimum representation for the keyword. For example, valid abbreviations for **TYPEptr** include **typ** and **type**.

The syntax of the compiler options varies according to the type of parameter that is used with the option. There are four types of parameters:

- Strings
- Filenames
- Switches
- Numbers

### Strings

If the option has a string parameter, and the string contains spaces, enclose the string with a pair of double quotation marks. For example, `/V"Version 1.0"` is correct. If there are no spaces in the string, the quotation marks are not necessary. For example, both `/VNew` and `/V"New"` are valid.

If the string itself contains double quotation marks, precede these with the backslash (`\`) character. For example, if the string is `abc"def`, specify it on the command line as `"abc\"def"`. This combination is the only escape sequence allowed within string options. Do not end a string with a backslash, as in `"abc\"`.

If the string is optional, do not put a space between the option and the string.

## Compiler Option Parameters

### Filenames

If you want to use a file that is in the current directory, specify only the filename. If the file you want to use is not in the current directory, specify the path and filename. For example, if your current directory is E:\, your source file is E:\myprog.c, and you compile using the defaults, your executable file will be called myprog.exe. If you want to put your executable file into the F:\ directory and call it newprog.exe, use the following command:

```
icc /FeF:\newprog.exe myprog.c
```

If you do not specify an extension for the executable file, .exe is assumed.

If your filename contains spaces (as permitted by the High Performance File System (HPFS) and the NT File System (NTFS)) or any elements of the HPFS or NTFS extended character set, it must be enclosed in double quotation marks. In such a case, do not put a space between the option and a filename or directory.

### Switches

Some options are used with plus (+) or minus (-) signs. If you do not use a sign, the compiler processes the option as if you had used the + sign. When you use an option that uses switches, you can combine the switches. For example, the following two option specifications have the same result:

```
/La+ /Le+ /Ls+ /Lx-  
/Laesx-
```

Note that the minus sign applies only to the switch immediately preceding it.

### Numbers

When an option uses a number as a parameter, do not put a space between the option and the number. When an option uses two numbers as parameters, separate the numbers with a comma. Do not leave a space between the numbers and the comma. The following option is specified correctly:

```
/Sg10,132
```

---

### Scope of Compiler Options

Options apply only to the source files that follow the option. The last, or rightmost, occurrence of these options is the one that is in effect for the source file or files that follow it.

In the following example, the file `module1.c` is compiled with the option `/L-` because this option follows `/L+`:

```
icc /L+ /L- module1.c
```

In the next example, the file `module1.c` is compiled with the `/L+` option, while `module2.c` is compiled with `/L-`:

```
icc /L+ module1.c /L- module2.c
```

**Exceptions**     The following options behave differently:

- `/D`     Defines a preprocessor macro. `/D` is different from other options in that the **first** definition of a macro is the one that is used. If a preprocessor macro is defined more than once, a warning appears.
- `/I`     Sets search paths for **#include** files. This option is cumulative. If you specify the option more than once, the parameters you specify are appended to the parameters previously stated. For example, the command
- ```
icc /Ia: /Ib:\cde /Ic:\fgh prog.c
```
- causes the following search path to be built:
- ```
a; ;b:\cde;c:\fgh
```
- `/B`     Passes options to the linker. Like `/I`, this option is cumulative. If you specify the option more than once, the parameters you specify are appended to the parameters previously stated. All options on the command line, and in environment variables, are accumulated **before** the object files are linked. The options apply to all object files linked.

### Specifying Options with Multiple Source Files

When you compile programs with multiple source files, an option applies to all the source files that follow it. For example, if you enter the following command:

```
icc /Oi+ main.c /Fa sub1.c /Lx /Oi- sub2.c
```

- The file `main.c` is compiled with the option `/Oi+`
- The file `sub1.c` is compiled with the options `/Oi+` and `/Fa+`
- The file `sub2.c` is compiled with the options `/Oi-`, `/Fa+` and `/Lx`

The name of the executable module will be the same as the name of the first source file (`main`), but with the extension `.exe`.

## Scope of Compiler Options

### ICC Combined with Options Entered on the Command Line

When you specify compiler options both in the ICC environment variable and on the command line, the compiler evaluates both sets of options. When the compiler is invoked:

1. The string associated with ICC is retrieved.
2. The command line is retrieved.
3. The command line is appended to the ICC string, combining the two into a single command line.
4. This combined command line is read from left to right, and the compiler option precedence rules are applied.
5. The files are compiled and linked using the options as interpreted in the previous step.

As a result, values in ICC are processed before the command line, and options on the command line override any conflicting options in ICC.

### Related Options

Some options are required with other options:

- If you specify the listing file option `/Le` (expand macros), or one of `/Li` or `/Lj` (expand **#include** files), you must also specify the `/Ls` option to include the source code.
- If you specify any of the listing options `/Lp` (set page length), `/Lt` (set title), or `/Lu` (set subtitle), you must also specify `/L` (produce listing file).
- If you specify `/Xs` (exclude specified files), you must also specify `/Ga` (turn on implicit SOM mode).

To use the Performance Analyzer, you must specify both the `/Gh` and `/Ti` options.

### Conflicting Options

Some options are incompatible with other options. If options specified on the command line are in conflict, the following rules apply:

- The syntax check option (`/Fc`) takes precedence over the output file generation (`/Fa`, `/Fb`, `/Fe`, `/Fm`, `/Fo`, and `/Ft`), intermediate code linker (`/Fw` and `/O1`), and preprocessor (`/P`, `/Pc`, `/Pd`, and `/Pe`) options.
- The preprocessor options (`/P`, `/Pc`, `/Pd`, and `/Pe`) take precedence over the output file generation (`/Fa`, `/Fb`, `/Fe`, `/Fl`, `/Fm`, `/Fo`, and `/Ft`), intermediate code linker (`/Fw`, `/Gu`, and `/O1`), precompiled header file (`/Fi` and `/Si`), and all listing file (`/L`) options.
- The option for no runtime environment (`/Rn`) takes precedence over the multithreading (`/Gm`), `ddnames` (`/Sh`), and memory file (`/Sv`) options.

## Scope of Compiler Options

- The option to not create an object file (/Fo-) takes precedence over the option to include debug information in the object (/Ti).
- The compile-only option (/C) takes precedence over the name executable module (/Fe) and generate linker map (/Fm) options.
- The no-optimization option (/O-) takes precedence over the instruction scheduler option (/Os+).
- The options to expand **#include** files in the listing (/Li and /Lj) take precedence over the precompiled header file options (/Fi and /Si).
- The option to expand user and system **#include** files (/Lj+) takes precedence over the option to expand user **#include** files only (/Li).
- The option to use the intermediate code linker (/Ol) requires some options to be consistently defined for all input files. The following options must have the same setting for all source files when you specify /Ol:

/G3	/G4	/G5
/Ge	/Gf	/Gh
/Gi	/Gr	/Gs
/Gw	/Nd	/O
/Oc	/Oi	/Om
/Op	/Os	/qtune
/qwin32s	/Re	/Rn
/Ti	/Tn	/W

 See “Using the Intermediate Code Linker” on page 81 for more information.

## Language-Dependent Options

Some VisualAge for C++ options are only valid when compiling C programs, while others only apply to C++ programs.



### C Programs Only

- /qalias Specifies aliasing rules. The typeptr, allptrs, and addrtaken suboptions can only be used in C programs. Only the ansi suboption can be used in C++ programs.
- /Sg Set margins for input files. This option is provided primarily for compatibility with IBM C/370. C++ does not require any such compatibility.
- /Sq Set sequence numbers for input files. This option is provided primarily for compatibility with IBM C/370. C++ does not require any such compatibility.
- /Sr Set type conversion rules. The C++ language only supports the new type conversion rules defined by the ANSI standard.
- /Ss Allow use of double slashes for comments. C++ allows double slashes to indicate comments as part of the language.

## Scope of Compiler Options

/S2 Allow only SAA Level 2 C constructs. There is no SAA definition of the C++ language.



### C++ Programs Only

/Fb Control generation of browser files.  
/Fr Give release order of class (SOM).  
/Fs Create and direct IDL file (SOM).  
/Ft Control generation of files for template resolution. The C language does not support templates.  
/Ga Turn on implicit SOM mode.  
/Gx Control inclusion of C++ exception-handling information. The C language does not include specific constructs for exception handling.  
/Gz Initialize SOM classes during static initialization.  
/Sc Allows constructs compatible with earlier versions of the C++ language. These constructs are not allowed in C.  
/Nx Set names of exception-handling segments.  
/qrtti Generate information for the typeid operator and the dynamic\_cast operator.  
/qsomvolattr Use volatile on attribute prototypes.  
/Xs Exclude files in specific directory when implicit SOM mode on.

## Compiler Options for Windows Programming

If you are using the VisualAge for C++ product to develop applications, you may need the following options:

Option	Description
/Se	Allow all VisualAge for C++ language extensions. (This is the default.)
/Gm	Use the multithread libraries.
/Gs-	Do not remove stack probes. (This is the default.)
/Wpro	Produce diagnostic messages about unprototyped functions. (These are generated by default).



## Examples of Compiler Options

### Examples of Compiler Options for Choosing Libraries

Figure 22 shows the combinations of compiler options you use to create a particular type of module, according to:

- Static or dynamic linking
- Threading level:
  - Single-thread (/Gm-)
  - Multithread (/Gm+)
- Library being used:
  - Standard (/Re)
  - Subsystem (/Rn)
- Module being built:
  - .exe file (/Ge+)
  - .dll file (/Ge-)

The defaults used by the compiler are:

- /Gd- (Use static linking)
- /Gm- (Use the single-thread library)
- /Re (Use the standard library)
- /Ge+ (Build an .exe file).

Figure 22. Combinations of Compiler Options for Specifying Libraries










Linking Type	Threading	Library used	Module Type	Options required in addition to defaults
Static	Single	Standard	EXE	None
Static	Single	Standard	DLL	/Ge-
Static	Multi	Standard	EXE	/Gm+
Static	Multi	Standard	DLL	/Gm+ /Ge-
Static	N/A	Subsystem	EXE	/Rn
Static	N/A	Subsystem	DLL	/Rn /Ge-
Dynamic	Single	Standard	EXE	/Gd+
Dynamic	Single	Standard	DLL	/Gd+ /Ge-
Dynamic	Multi	Standard	EXE	/Gd+ /Gm+
Dynamic	Multi	Standard	DLL	/Gd+ /Gm+ /Ge-
Dynamic	N/A	Subsystem	EXE	/Gd+ /Rn
Dynamic	N/A	Subsystem	DLL	/Gd+ /Rn /Ge-

## Compiler Option Classification

---

### Compiler Option Classification

The compiler options are divided into groups by function. The following list tells you which options are in each group.

-  “Output File Management Options” on page 121  
/F
-  “**#include** File Search Options” on page 128  
/I /Xc /Xi
-  “Listing File Options” on page 130  
/L
-  “Debugging and Diagnostic Information Options” on page 135  
/N /qdbgunref /Ti /Tm /Tn /Tx /W
-  “Source Code Options” on page 143  
/J /S /qbitfields /qdigraph /qlonglong /Tc /Td /Tp
-  “Preprocessor Options” on page 151  
/D /P /U
-  “Code Generation Options” on page 154  
/G /M /Nd /Nt /Nx /O /qalias /qisolated\_call /qtune /R
-  “Other Options” on page 171  
/B /C /H /Q /qautoimported /qautothread /qignprag  
/qlibansi /qmakedep /qro /qsomvolattr /qwin32s /V
-  “Options for New ANSI Standards” on page 176  
/qrtti

The table that follows gives all options, in all groups, in alphabetical order. The options are described in more detail in the sections following the table.

## Compiler Options Summary

### Compiler Options Summary

Figure 23 (Page 1 of 5). Compiler Options Summary

Option	Description	Default	Page
/?	Display list of compiler options with descriptions.	None.	171
/B"options"	Pass options to linker, in addition to default options.	/B""	172
/C[+ -]	Perform compile without linking, instead of compiling and linking.	/C-	172
/Dname /Dname=[n]	Define preprocessor macros.	None.	151
/Fa[+ -][dir][name]	Produce, name, and direct assembler listing file.	/Fa-	122
/Fb[+ -]*]	Produce a browser file.	/Fb-	123
/Fc[+ -]	Perform syntax check only, instead of a full compile.	/Fc-	123
/Fename	Specify name of executable output file.	Name of first source file	124
/Fi[+ -][dir][name]	Produce, name, and direct precompiled header file.	/Fi-	124
/Fl[+ -][dir][name]	Produce, name, and direct listing file.	/Fl-	125
/Fm[+ -] /Fmname	Produce and name linker map file.	/Fm-	125
/Fo[+ -][dir][name]	Control and name object file.	/Fo[+]	126
/Fr<classname>	Give release order of class.	None	170
/Fs[+ -][name][dir]	Create and direct IDL file.	/Fs-	170
/Ft[+ -][dir]	Control and direct files for template resolution.	/Ft[+]	126
/Fw[+ -][dir][name]	Create intermediate code files only, instead of a full compilation. Specify name and directory for files.	/Fw-	127
/G<3 4 5>	Specify the processor for which the code is optimized.	/qtune=blend	154
/Ga[+ -]	Turn on implicit SOM mode.	/Ga-	168
/Gb[+ -]	Disable direct access to attributes for DSOM.	/Gb-	169
/Gd[+ -]	Dynamically link to the runtime library, instead of linking statically.	/Gd-	155
/Ge[+ -]	Build .exe or .dll file.	/Ge[+]	155
/Gf[+ -]	Use fast floating-point execution.	/Gf+	156
/Gh[+ -]	Enable code for performance analysis.	/Gh-	156
/Gi[+ -]	Use fast integer execution.	/Gi-	157
/Gl[+ -]	Remove unreferenced functions.	/Gl-	157
/Gm[+ -]	Link with the multithread library, instead of the single-thread library.	/Gm-	158
/Gn[+ -]	Hide default library information from linker.	/Gn-	158

## Compiler Options Summary

Figure 23 (Page 2 of 5). Compiler Options Summary

Option	Description	Default	Page
/Gs[+ -]	Remove stack probes.	/Gs-	158
/Gu[+ -]	Stop external functions from using data defined in intermediate files.	/Gu-	159
/Gw[+ -]	Generate FWAIT instruction after each floating-point load instruction.	/Gw-	159
/Gx[+ -]	Remove C++ exception handling information.	/Gx-	160
/Gz[+ -]	Do not prebuild SOM classes at static initialization time.	/Gz-	169
/Hnum	Set maximum length of external names.	/H255	172
/Ipath[:path]	Specify <b>#include</b> search paths, in addition to directory of source file and paths in INCLUDE.	No additional paths.	128
/J[+ -]	Treat unspecified char variables as signed char, instead of unsigned char.	/J[+]	143
/L[+ -]	Produce a minimal listing file.	/L-	131
/La[+ -]	Include a minimal layout in the listing file.	/La-	131
/Lb[+ -]	Include a layout in the listing file.	/Lb-	132
/Le[+ -]	Expand macros in the listing file.	/Le-	132
/Lf[+ -]	Set all listing options on.	/Lf-	132
/Li[+ -]	Expand user <b>#include</b> files in the listing file.	/Li-	133
/Lj[+ -]	Expand user and system <b>#include</b> files in the listing file.	/Lj-	133
/Lpnum	Set page length of listing file.	/Lp66	133
/Ls[+ -]	Include the source code in the listing file.	/Ls-	134
/Lt"string"	Set title string for the listing file.	Name of first source file.	134
/Lu"string"	Set subtitle string in the listing file.	/Lu""	134
/Lx[+ -]	Generate a minimal cross-reference table in the listing file.	/Lx-	135
/Ly[+ -]	Generate a cross-reference table in the listing file.	/Ly-	135
/M<p s c t>	Set default calling convention.	/Mp	160
/Nn	End compilation when error count reaches <i>n</i> .	No limit.	135
/Ndname	Set names of default data, uninitialized data, and constant segments.	Use DATA32, BSS32, and CONST32_RO.	161
/Ntname	Set names of default code or text segment.	Use CODE32.	161
/Nxname	Set names of exception-handling segments.	EH_CODE and EH_DATA	162
/O[+ -]2 3]	Optimize code.	/O-	162
/Oc[+ -]	Optimize code for size.	/Oc-	163

## Compiler Options Summary

Figure 23 (Page 3 of 5). Compiler Options Summary

Option	Description	Default	Page
/Oi[+ -] /Oi[ <i>value</i> ]	Inline specified user functions.	/Oi- /Oi+ when /O+ /Oi0	163
/Ol[+ -]	Use intermediate code linker.	/Ol-	164
/Om[+ -]	Limit working set size.	/Om-	164
/Op[+ -]	Do not perform optimizations that involve the stack pointer.	/Op+	164
/Os[+ -]	Invoke the instruction scheduler.	/Os- /Os+ when /O+	165
/P[+ -]	Run the preprocessor only, instead of a full compile.	/P-	152
/Pc[+ -]	Preserve source code comments in preprocessor output.	/Pc-	152
/Pd[+ -]	Redirect preprocessor output to stdout.	/Pd-	152
/Pe[+ -]	Suppress #line directives in preprocessor output.	/Pe-	153
/Q[+ -]	Suppress the compiler logo when invoking the compiler.	/Q-	173
/qalias	Specify which aliasing rules can be used during optimization.	None.	165
/qautoimported /qnoautoimported	Import all external references.	/qnoautoimported	173
/qautothread /qnoautothread	Turn static, nonconstant data into thread-local data.	/qnoautothread	173
/qbitfields = <i>&lt;signed unsigned&gt;</i>	Specify default sign of bitfields.	/qbitfields=unsigned	143
/qdbgunref /qnodbgunref	Use referenced and unreferenced symbols during debugging.	/qnodbgunref	136
/qdigraph /qnodigraph	Permit digraph and keyword operators.	/qnodigraph	143
/qignprag= <i>&lt;disjoint  isolated all&gt;</i>	Ignore references to #pragma disjoint and #pragma isolated_call.	None.	174
/qisolated_call = <i>fname[:fname]...</i>	List functions that do not change data objects at time of function call.	None.	165
/qlibansi /qnolibansi	Process ANSI C library names as system functions.	/qnolibansi	174
/qlonglong /qnolonglong	Process or disallow data type of LONG LONG INT.	/qlonglong	144
/qmakedep	Create dependency files to include in makefile.	None.	174
/qro /qnoro	Use read-only, or not read-only (read/write) storage for string literals.	/qro	175
/qrtti= <i>rtoption</i> /qnortti	Generate information for the typeid operator and the dynamic_cast operator.	/qnortti	176

## Compiler Options Summary

Figure 23 (Page 4 of 5). Compiler Options Summary

Option	Description	Default	Page
/qsomvolattr /qnosomvolattr	Use volatile on attribute prototypes.	/qnosomvolattr	175
/qtune= <i>option</i>	Specify the processor for which code is optimized.	/qtune=blend	166
/qwin32s /qnowin32s	Help port your program to the Win32s runtime environment.	/qwin32s	166
/R[e n]	Generate code that can be used as a subsystem without a runtime environment.	/Re	167
/S<a c e 2>	Set language level.	/Se	144
/Sd[+ -]	Set the default file extension for source files to .c, instead of .obj.	/Sd-	144
/Sg[l][,<r *>] /Sg[-]	Set left and right margins for the input file, and ignore text outside these margins.	/Sg-	145
/Sh[+ -]	Allow use of ddnames.	/Sh-	146
/Si[+ -] /Si[ <i>dir</i> ][ <i>name</i> ]	Use precompiled header files, if they exist and are current.	/Si-	146
/Sm[+ -]	Ignore obsolete keywords, instead of treating them like any other identifier.	/Sm-	147
/Sn[+ -]	Allow use of DBCS.	/Sn-	147
/Sp[1 2 4 8 16]	Specify alignment or packing of data items within structures and unions.	/Sp8	147
/Sq[l][,<r *>] /Sq[-]	Ignore text in specified columns, instead of processing all the contents of the input file.	/Sq-	148
/Sr[+ -]	Use old-style rules for type conversion, instead of new-style rules.	/Sr-	148
/Ss[+ -]	Allow double slashes to indicate comments.	/Ss+	148
/Su[+ -]1 2 4	Control size of enum variables, instead of using the SAA rules.	/Su-	149
/Sv[+ -]	Allow use of memory files.	/Sv-	149
/Tc <i>filename</i>	Compile the following file as a C source file, regardless of its extension.	Compile based on file extension.	149
/Td[c p]	Specify the default language (C or C++) for files, instead of compiling according to the file extension.	/Td	150
/Ti[+ -]	Generate debugger information.	/Ti-	136
/Tm[+ -]	Enable debug version of memory management functions.	/Tm-	137
/Tn[+ -]	Generate partial debugger information.	/Tn-	137
/Tp <i>filename</i>	Compile the following file as a C++ source file, regardless of its extension.	Compile based on file extension.	150

Figure 23 (Page 5 of 5). Compiler Options Summary


Option	Description	Default	Page
/Tx[+ -]	Provide a complete machine-state dump when an exception occurs, instead of providing only the exception message and address.	/Tx-	138
/U<name *>	Undefine macros.	Retain macros.	153
/V"string"	Include a version string in the object and executable files.	No string.	175
/W<0 1 2 3>	Set severity level of messages the compiler produces and counts.	/W3	138
/Wgrp[+ -][grp]	Generate or suppress messages in the <i>grp</i> group.	/Wall-pro+ret+cnd+	138
/Xc[+ -]	Do not search paths specified using /I.	/Xc-	129
/Xi[+ -]	Do not search paths specified in INCLUDE.	/Xi-	129
/Xs[dir -]	Exclude files in directory <i>dir</i> when /Ga is on (implicit SOM mode).	/Xs-	169

## Output File Management Options

Use these options to control the files that the compiler produces.


**Note:** You do not need the plus symbol (+) when specifying an option: the forms /L+ and /L are equivalent.

### Filenames and Extensions

If you do not specify an extension for the file management options that take a filename as a parameter, the default extension is used. For example, if you specify /Flcome, the listing file will be called come.lst. Although you can specify an extension of your own choosing, you should use the default extensions.  See “File Types” on page 64 for more information on default extensions.

If you use an option without using an optional *name* parameter, the name of the following source file and the default extension is used, with the exception of the /Fm option. If you do not specify a name with /Fm, the name of the first file given on the command line is used, with the default extension .map.

**Note:** If you use the /Fe option, you **must** specify a name or a path for the file. If you specify only a path, the file will have the same name as the first source file on the command line, with the path specified.

 See “Filenames” on page 110 for more information on using filenames as parameters with options.

## /Fa Option

### Examples



- Perform syntax check only:

```
icc /Fc+ myprog.c
```

- Name the object file:

```
icc /Fobarney.obj fred.c
```

This names the object file barney.obj instead of the default, fred.obj.

- Name the executable file:

```
icc /Febarney.exe fred.c
```

This names the object file barney.exe instead of the default, fred.exe.

- Name the listing file:

```
icc /Floutput.my /L fred.c
```

This creates a listing output file called output.my instead of fred.lst.

- Name the linker map file:

```
icc /Fmoutput.map fred.c
```

This creates a linker map file called output.map instead of fred.map.

- Name the assembler listing file:

```
icc /Fabarney fred.c
```

This names the output barney.asm instead of fred.asm.

## /Fa

### Syntax:

```
/Fa[+|-]  
/Fa[dir][name]
```

### Default:

```
/Fa-
```

Use /Fa to produce, name, and direct an assembler listing file that has the source code as comments. The listing file will be *name.asm* and will be placed in directory *dir*.

The compiler produces a listing file for each source file that follows the option on the command line. The name you provide applies only to the first listing file.

If you do not specify a name or directory, then the listing takes the same name as the source file, with the extension .asm, and is put in the current directory.



## /Fb Option •/Fc Option

If the directory you specify is not valid, the compiler does not generate a listing file: it generates a warning message and the option does not take effect.

**Note:** The listing is not guaranteed to compile.

By default, the compiler does not create an assembler listing file.


### /Fb

**Syntax:**  
/Fb[+|-]\*]


**Default:**  
/Fb-



Use /Fb to produce a browser file, for use by the VisualAge for C++ Browser. The file has the same name as the next source file with the extension .pdb. You can include maximum information in the browser file by specifying /Fb\*. You do not need to specify /Fb\* unless the compiler issues a message that tells you to use the option.

 See “Creating Files to Use with the Browser” on page 335 for more information on the differences between /Fb and /Fb\*.

If you are compiling and linking in one step, the compiler passes the /BROWSE option to the linker. If you are compiling only, you must specify the /BROWSE option directly, when you link, to preserve the browse information.

The browser file allows the VisualAge for C++ browser to browse your program.  See Part 6, “Browsing Programs and Libraries” on page 327 for more information on the browser.

**Note:** This option is valid for C++ files only.

By default, the compiler does not produce a browser file.

### /Fc

**Syntax:**  
/Fc[+|-]

**Default:**  
/Fc-

Use /Fc to perform only a syntax check. The only output files you can produce when this option is in effect are listing (.lst) files.

By default, the compiler compiles and produces output files according to any other options in effect.

## /Fe Option •/Fi Option

### /Fe

**Syntax:**

/Fename

**Default:**

Use name of first source file, and add the .exe or .dll extension.

Use /Fe to specify the name (and optional path) of the .exe or .dll file you are producing. The executable output file will be *name.exe* or *name.dll*.

If you do not provide a name, the file takes the same name as the first source file, with the extension .exe or .dll. The output file is placed in the current directory.

### /Fi

**Syntax:**

/Fi[+|-]  
/Fi[dir][name]

**Default:**

/Fi-

Use /Fi to control creation of precompiled header files. The compiler creates a precompiled header file if none exists or if the existing one is out-of-date.


If you specify a *name* or *directory* with the option, then the precompiled headers are placed in a file with the name and in the directory you specify.

You can also use the **#pragma hdrfile** directive to tell the compiler what file to generate. You must still specify /Fi.

If you do not specify a name or directory, the file is named csetc.pch (if the next source file is a C file) or csetcpp.pch (if the next source file is a C++ file), and placed in the current working directory.

Use the /Si option to use the precompiled header files. Use /Fi and /Si in combination to ensure that your precompiled header files are always up to date.

**Note:** The file you generate (/Fi) must be the same file you use (/Si). If you specify different filenames or directories with the two options, the name or directory specified last is used with both options. If you specify a filename or directory with **#pragma hdrfile**, it overrides the name or directory specified with the options.

 See “Using Precompiled Headers” on page 94 for more information.

By default, the compiler does not create a precompiled header file.

## /Fl Option •/Fm Option

### /Fl

**Syntax:**


/Fl [+|-]  
/Fl [*dir*] [*name*]

**Default:**

/Fl -

Use /Fl to produce, name, and direct a listing file. The listing file will be *name.lst*, and will be placed in directory *dir*.

The compiler produces a separate listing file for each source file that follows the option on the command line. The name you provide applies only to the first listing file.

 See “Compiler Listings” on page 79 for more information.

If you do not specify a name or directory, the listing takes the same filename as the source file, with the extension *.lst*, and is put in the current directory.

If the directory you specify is not valid, the compiler does not generate a listing file: it generates a warning message and the option does not take effect.

By default, the compiler does not produce a listing file.

### /Fm

**Syntax:**


/Fm [+|-]  
/Fm *name*

**Default:**

/Fm -

Use /Fm to produce and name a linker map file. The map file will be *name.map*.

If you do not provide a name, the map file takes the same filename as the source file, with the extension *.map*.

 See “Generating a Map File” on page 191 for more information.

By default, the compiler does not produce a map file.

## /Fo Option •/Ft Option

### /Fo

**Syntax:**


/Fo[+|-]  
/Fo[*dir*][*name*]

**Default:**

/Fo[+]

Use /Fo to produce, name, and direct an object file. The object file will be *name.obj*, and will be placed in directory *dir*.

The compiler produces a separate object file for each source file that follows the option on the command line. The name you provide applies only to the first object file.

 See “Object Files” on page 76 for more information.

If you do not specify a name or a directory, the object file takes the same filename as the source file, with the extension *.obj*, and is put in the current directory.

If the directory you specify is not valid, the compiler generates a warning message, and places the object file in the current directory, with its default name.

By default, the compiler produces an object file with the same name as the source file and the extension *.obj*.

Specify /Fo- if you do not want the compiler to create an object file.

### /Ft

**Syntax:**

/Ft[+|-]  
/Ft*dir*

**Default:**

/Ft[+]



Use /Ft to control generation of files for template resolution.

**Note:** This option is valid for C++ files only. The C language does not support the use of templates.

Specify /Ft- to suppress generation of files for template resolution.

Specify /Ft*dir* to generate the files for template resolution and place them in the *dir* directory.

By default, files for template resolution are generated and stored in the TEMPINC subdirectory under the current directory.

## /Fw Option

### /Fw

**Syntax:**


/Fw[+|-]  
/Fw[*dir*][*name*]

**Default:**

/Fw-

Use /Fw to produce, name, and direct intermediate code files, without completing compilation. The intermediate code files will be *name.w*, *name.wh*, *name.wi*, and *name.ws*, and will be placed in directory *dir*.

The compiler produces a separate set of intermediate code files for each source file that follows the option on the command line. The name you provide applies only to the first set of intermediate code files.

Intermediate code files are used by the intermediate code linker. For more information,  see “Using the Intermediate Code Linker” on page 81.

If you do not specify a name, the intermediate code files take the same filename as the source file, with the extensions *.w*, *.wh*, *.wi*, and *.ws*.

If the directory you specify is not valid, the compiler does not save the intermediate code files: it generates a warning message and the option does not take effect.

By default, the compiler performs regular compilation, without saving intermediate code files.

## /I Option

---

### #include File Search Options

Use these options to control which paths are searched when the compiler looks for **#include** files. The paths that are searched are the result of the information in the INCLUDE and ICC environment variables, combined with how you use the following compiler options.

#### Using the #include File Search Options

The /I option must be followed by one or more directory names. A space may be included between /I and the directory name. If you specify more than one directory, separate the directory names with a semicolon.

If you use the /I option more than once, the directories you specify are appended to the directories you previously specified. For example:


```
/Id:\hdr;e:\ /I f:\
```

is equivalent to

```
/Id:\hdr;e:\;f:\;
```

If you specify search paths using /I in both the ICC environment variable and on the command line, **all** the paths are searched. The paths specified in ICC are searched before those specified on the command line.

Once you use the /Xc option, the paths previously specified by using /I cannot be recovered. You have to use the /I option again if you want to reuse the paths canceled by /Xc.

The /Xi option has no effect on the /Xc and /I options. For further information on **#include** files and search paths,  see “Controlling #include Search Paths” on page 68.

## /I

#### Syntax:

```
/Ipath[;path]
```

#### Default:

Directory of source file, paths in  
INCLUDE environment variable

Use /I to specify **#include** search path(s). The compiler will search *path[;path]*. Note that for user include files, the directory of the source file is always searched first.

## **/Xc Option •/Xi Option**

By default, the compiler searches the directory of the source file (for user files only), and then search paths given in the INCLUDE environment variable.

### **/Xc**

**Syntax:**

/Xc[+|-]

**Default:**

/Xc-

Use /Xc to stop the compiler from searching paths specified using /I.

By default, the compiler searches paths specified using /I.

### **/Xi**

**Syntax:**

/Xi[+|-]

**Default:**

/Xi-

Use /Xi to stop the compiler from searching paths specified in the INCLUDE environment variable.

By default, the compiler searches paths specified in the INCLUDE environment variable.

---

## Listing File Options

Use these options to control whether or not a listing file is produced, the type of information in the listing, and its appearance.

**Note:** The following options only modify the appearance of a listing; they do not cause a listing to be produced. Use them with one of the other listing file options, or with the /F1 option, to produce a listing:

/Le /Li /Lj /Lp /Lt /Lu

If you specify any of the /Le, /Li, or /Lj options, you must also specify the /L, /Lf, or /Ls option.

### Including Information about Your Source Program

You can use three options to include information about your source program in the listing file:

/Ls[+]	Includes your source program in the listing file.
/Li[+]	Shows the included text after the user <b>#include</b> directives.
/Lj[+]	Shows the included text after both user and system <b>#include</b> directives.

See the option descriptions for additional information.

**Note:** If you specify the /Lj option, /Li[+] and /Li- have no effect.

### Including Information about Variables

The options that produce information about the variables used in your program provide the following amount of detail:

/La[+]	Includes a table of all the referenced struct and union variables in the source program.
/Lb[+]	Includes a table of all struct and union variables in the program.
/Le[+]	Includes all expanded macros in the listing file.
/Lx[+]	Includes a cross-reference table that contains a list of the referenced identifiers in the source file together with the numbers of the lines in which they appear.



## **/L Option •/La Option**

**/Ly[+]** Includes a cross-reference table that contains a list of all identifiers referenced by the user and all external identifiers, together with the numbers of the lines in which they appear.


See the option descriptions for additional information.

### **/L**

**Syntax:**  
**/L[+|-]**

**Default:**  
**/L-**

Use **/L** to produce a listing file. The listing file contains only a prolog and error messages. You can modify the contents of the listing using other listing file options.

 See “Compiler Listings” on page 79 for more information.

By default, the compiler does not produce a listing file.

### **/La**

**Syntax:**  
**/La[+|-]**

**Default:**  
**/La-**

Use **/La** to include a table in the listing file that shows the referenced struct and union variables in the source program. The table shows how each structure and union in the program is mapped. It contains the following information:

- The name of the structure or union and the elements within each
- The byte offset of each element from the beginning of the structure or union. The bit offset for unaligned bit data is also given.
- The length of each element
- The total length of each structure, union, and substructure in both packed and unpacked formats

By default, the listing file does not include a layout.

## **/Lb Option •/Lf Option**

### **/Lb**

**Syntax:**

/Lb[+|-]

**Default:**

/Lb-

Use /Lb to include a table in the listing file that shows all the struct and union variables in the source program. The table shows how each structure and union in the program is mapped. It contains the following information:

- The name of the structure or union and the elements within each
- The byte offset of each element from the beginning of the structure or union. The bit offset for unaligned bit data is also given.
- The length of each element
- The total length of each structure, union, and substructure in both packed and unpacked formats

By default, the listing file does not include a layout.

### **/Le**

**Syntax:**

/Le[+|-]

**Default:**

/Le-

Use /Le to expand all macros in the listing file.

**Note:** To use /Le, you must also specify either /L or /Ls.

By default, the listing file does not show macros expanded.

### **/Lf**

**Syntax:**

/Lf[+|-]

**Default:**

/Lf-

Use /Lf to set all listing options on, and generate a listing file.

By default, all listing options are off.

## /Li Option •/Lp Option

### /Li

**Syntax:**  
/Li [+|-]

**Default:**  
/Li-

Use /Li to expand user **#include** files in the listing file.

**Note:** To use /Li, you must also specify either /L or /Ls.

By default, the listing file does not show user **#include** files expanded.

### /Lj

**Syntax:**  
/Lj [+|-]

**Default:**  
/Lj-

Use /Lj to expand user and system **#include** files in the listing file.

If you use HPFS and have very long filenames, there may not be enough room for the filenames on the lines showing the included code. Counters are used in the INCLUDE column of the listing output, and the filename corresponding to each number is given at the bottom of the source listing.

**Note:** To use /Lj, you must also specify either /L or /Ls.

By default, the listing file does not show user and system **#include** files expanded.

### /Lp

**Syntax:**  
/Lpnum

**Default:**  
/Lp66

Use /Lp to set the page length in the listing file. Each page will be *num* lines long. You can set *num* to any number from 15 to 65535.

By default, the listing file has 66 lines per page.

## **/Ls Option •/Lu Option**

### **/Ls**

**Syntax:**  
/Ls[+|-]

**Default:**  
/Ls-

Use /Ls to include the source code in the listing file.

By default, the listing file does not include the source code.

### **/Lt**

**Syntax:**  
/Lt"*string*"

**Default:**  
Use name of first source file

Use /Lt to set the title string of the listing file to *string*. Maximum string length is 256 characters.

By default, the title string is set to the name of the first source file.

**Note:** You can also specify a title using the **#pragma title** directive, but this title does not appear on the first page of the listing output.

### **/Lu**

**Syntax:**  
/Lu"*string*"

**Default:**  
/Lu""

Use /Lu to set the subtitle string in the listing file to *string*. Maximum string length is 256 characters.

By default, no subtitle is set (null string).

**Note:** You can also specify a subtitle using the **#pragma subtitle** directive, but this subtitle does not appear on the first page of the listing output.

## /Lx Option • /N Option

### /Lx

**Syntax:**  
/Lx[+|-]

**Default:**  
/Lx-

Use /Lx to generate a cross-reference table in the listing file for referenced variable, structure, and function names that shows line numbers where names are declared.

By default, the listing file does not include the cross-reference table.

### /Ly

**Syntax:**  
/Ly[+|-]

**Default:**  
/Ly-

Use /Ly to generate a cross-reference table in the listing file of all variable, structure, and function names, plus all local variables referenced by the user.

By default, the listing file does not include the cross-reference table.

---

## Debugging and Diagnostic Information Options

Use these options to help debug your programs.

Use /Ti to prepare your output for debugging with the VisualAge for C++ Debugger.

Use /Wgrp to control what types of diagnostic messages are produced.

**Note:** The information generated by the VisualAge for C++ Debugger and /Wgrp options is provided to help you diagnose problems in your code. Do not use the diagnostic information as a programming interface.

### /N

**Syntax:**  
/Nn

**Default:**  
No limit

Use /N to set the maximum number of errors before compilation aborts. Compilation ends when the error count reaches *n*.

By default, the compiler sets no limit on the number of errors.

## **/qdbgunref Option •/Ti Option**

### **/qdbgunref**

**Syntax:**

/qdbgunref  
/qnodbgunref

**Default:**

/qnodbgunref

Use /qdbgunref to generate symbol table information for unreferenced variables.

By default, such information is not generated for unreferenced variables. This reduces the size of an executable compiled with the /Ti option.

### **/Ti**

**Syntax:**

/Ti [+|-]

**Default:**

/Ti-

Use /Ti to generate information for the VisualAge for C++ Debugger and for the Performance Analyzer.

By default, the compiler does not generate debug information. When you use /Ti+, do not turn on optimization (/O+, /Oc+, /Oi+, or /Os+), unless you are using the information with the Performance Analyzer, and not with the Debugger. Because the compiler produces debugging information as if the code were not optimized, the information may not accurately describe an optimized program being debugged, which makes debugging difficult. Accurate symbol and type information is not always available.

If you cannot avoid debugging an optimized program, turn the scheduler off (/Os-), and step through the program at the assembly level, using the **Register** and **Storage** windows for information.

To make full use of the VisualAge for C++ Debugger, set optimization off (the default).



For more information on the VisualAge for C++ Debugger, see Part 5, “Debugging Your Program” on page 261.

For more information on Performance Analyzer, see Part 7, “Performance Execution Trace Analyzer” on page 429.

## /Tm Option •/Tn Option


### /Tm

**Syntax:**

/Tm[+|-]

**Default:**

/Tm-

Use /Tm to enable debug versions of memory management functions. The debug memory management functions (`_debug_calloc`, `_debug_malloc`, `new`, and so on) are then used in place of the regular memory management functions. This option defines the `__DEBUG_ALLOC__` macro.  See the *C Library Reference* for information on the C debug memory management functions and the *Language Reference* for information on the debug versions of `new` and `delete`.

When you specify /Tm, the compiler generates additional code at the beginning of every function that preinitializes the local variables for the function. This makes it easier to find uninitialized local variables.

By default, the compiler uses the regular memory management functions (`calloc`, `malloc`, `new`, and so on), and does not preinitialize their local storage.

### /Tn

**Syntax:**

/Tn[+|-]

**Default:**

/Tn-

Use /Tn to generate abbreviated information for the debugger. You can then use the debugger to single-step through the source view of the files, but you cannot view variables.

Specify /Ti to generate more complete debugger information.

If you specify both /Tn and /Ti, the compiler will generate the more complete debugging information indicated by /Ti.

By default, the compiler does not generate line number information.

## **/Tx Option •/Wgrp Option**

### **/Tx**

**Syntax:**

/Tx[+|-]

**Default:**

/Tx-

Use /Tx to provide a complete machine-state dump when an exception occurs.

By default, the compiler provides only the exception message and address when an exception occurs, and does not provide a complete machine-state dump.


### **/W**

**Syntax:**

/W<0|1|2|3>

**Default:**

/W3

Use /W to set the severity level of messages the compiler produces, which causes the error count to increment.  See the online *User's Guide* for a description of error messages and severity levels.

By default (/W3)), the compiler produces and counts all message types (severe error, error, warning, and informational).

You can set the following severity levels:

/W0 Produce and count only severe errors.

/W1 Produce and count severe errors and errors.

/W2 Produce and count severe errors, errors, and warnings.

/W3 Produce and count all message types (severe error, error, warning, and informational).

### **/Wgrp**

**Syntax:**

/Wgrp[+|-] [grp]

**Default:**

/Wall-pro+ret+cnd+

Use /Wgrp to generate messages in the *grp* group. You can specify more than one group.

The /Wgrp options control informational messages that warn of possible programming errors, weak programming style, and other information about the structure of your programs. Similar messages are in groups, or suboptions, to give you greater control



## **/Wgrp Option**

over which types of messages you want to generate. You can also specify these groups in your source code, with **#pragma info**.

By default, the compiler generates diagnostic messages in the `pro`, `ret`, and `end` groups.

When you specify `/Wall[+]`, all suboptions are turned on and all possible diagnostic messages are reported. Because even a simple program that contains no errors can produce many informational messages, you may not want to use `/Wall` very often. You can use the suboptions alone or in combination to specify the type of messages that you want the compiler to report. Suboptions can be separated by an optional `+` sign. To turn off a suboption, you must place a `-` sign after it.

You can also combine the `/W[0|1|2|3]` options with the `/Wgrp` options.

## /Wgrp Option

The following table lists the message groups and the message numbers that each controls. Messages generated for C files begin with EDC0, while messages for C++ files begin with EDC3.


Figure 24 (Page 1 of 2). /Wgrp Options

<i>grp</i>	<b>Controls Messages About ...</b>	<b>Messages</b>
/Wall	All diagnostics.	All message numbers listed in this table.
/Wcls	Use of classes.	EDC3110, EDC3253, EDC3266
/Wcmp	Possible redundancies in unsigned comparisons.	EDC3138
/Wcnd	Possible redundancies or problems in conditional expressions.	EDC0424, EDC0425, EDC0426, EDC0427, EDC0420, EDC0421, EDC0422, EDC0423, EDC3107, EDC3130, EDC3388, EDC3389, EDC3390, EDC3391, EDC3392, EDC3393
/Wcns	Operations involving constants.	EDC0475, EDC0476, EDC0477, EDC3131, EDC3219, EDC3220
/Wcnv	Conversions.	EDC3313, EDC3528
/Wcpy	Problems generating copy constructors.	EDC3199, EDC3200
/Wdcl	Consistency of declarations.	EDC0716
/Weff	Statements with no effect.	EDC0509, EDC0435, EDC0436, EDC0437, EDC0473, EDC0474, EDC0478, EDC0479, EDC0483, EDC3165, EDC3215
/Wenu	Consistency of enum variables.	EDC0439, EDC0440, EDC0471, EDC3137, EDC3366
/Wext	Unused external definitions.	EDC0415, EDC0493, EDC0494, EDC3127
/Wgen	General diagnostics.	EDC0438, EDC0448, EDC0466, EDC0480, EDC0489, EDC0492, EDC3101
/Wgnr	Generation of temporary variables.	EDC3151
/Wgot	Usage of goto statements.	EDC0413
/Wini	Possible problems with initialization.	EDC0444, EDC0445, EDC0446, EDC0447, EDC0482
/Winl	Functions not inlined.	EDC3542
/Wlan	Effects of the language level.	EDC3116
/Wobs	Features that are obsolete.	EDC0450, EDC0470
/Word	Unspecified order of evaluation.	EDC0428, EDC0429, EDC0430, EDC0431, EDC0432
/Wpar	Unused parameters.	EDC0414, EDC3126

## /Wgrp Option

Figure 24 (Page 2 of 2). /Wgrp Options

<i>grp</i>	<b>Controls Messages About ...</b>	<b>Messages</b>
/Wpor	Nonportable language constructs.	EDC0433, EDC0434 EDC3108, EDC3133, EDC3135, EDC3136, EDC3307
/Wppc	Possible problems with using the preprocessor.	EDC0076, EDC0290, EDC0293, EDC0311, EDC0312, EDC0313, EDC0389, EDC0441, EDC0442, EDC0443, EDC0457, EDC0468
/Wppt	Trace of preprocessor actions.	EDC0467
/Wpro	Missing function prototypes.	EDC0304
/Wrea	Code that cannot be reached.	EDC0472, EDC0520, EDC3119
/Wret	Consistency of return statements.	EDC0449, EDC0481
/Wtrd	Possible truncation or loss of data or precision.	EDC0374, EDC0416, EDC0418, EDC0419, EDC0451, EDC0452, EDC0453, EDC0495, EDC3108, EDC3135, EDC3136
/Wtru	Variable names truncated by the compiler.	EDC0484
/Wund	Casting of pointers to or from an undefined class.	EDC3098, EDC3397, EDC3405, EDC3406
/Wuni	Uninitialized variables.	EDC0412
/Wuse	Unused auto and static variables.	EDC0409, EDC0410, EDC0469, EDC0490, EDC0491, EDC3002, EDC3099, EDC3100, EDC3101
/Wvft	Generation of virtual function tables.	EDC3280, EDC3281, EDC3282

 More information about the messages generated by the */Wgrp* options is available in the online version of the *User's Guide*.

## /Wgrp Option

### Examples



- Produce all diagnostic messages:

```
icc /Wall blue.c
icc /Wall+ blue.c
```

- Produce diagnostic messages about:

- Unreferenced parameters
- Missing function prototypes
- Uninitialized variables

by turning on the appropriate suboptions:

```
icc /Wpar+pro+uni blue.c
icc /Wparprouni blue.c
```

- Produce all diagnostic messages except:

- Warnings about assignments that can cause a loss of precision
- Preprocessor trace messages
- External variable warnings

by turning **on** all options, and then turning **off** the ones you do not want:

```
icc /Wall+trd-ppt-ext- blue.c
```

- Produce only basic diagnostics, with all other suboptions turned off:

```
icc /Wgen+ blue.c
```

- Produce only basic diagnostics and messages about severe errors, errors, or warnings (/W2):

```
icc /Wgen2 blue.c
```

---

## Source Code Options

Use these options to control how the VisualAge for C++ compiler interprets your source file. This control is especially useful, for example, if you are migrating code or ensuring consistency with a particular language standard.

### /J

**Syntax:**

/J[+|-]

**Default:**

/J+

Use /J- to treat unspecified char variables as signed char, for arithmetic and compare operations.

This option does **not** change the definition of the variable. In C++, char, signed char, and unsigned char are three distinct types. Unspecified char variables are considered signed char or unsigned char for arithmetic and compare operations only. You cannot convert one type to another type without casting.

By default (/J[+]), unspecified char variables are treated as unsigned char.

### /qbitfields

**Syntax:**

/qbitfields=<signed|unsigned>

**Default:**

/qbitfields=unsigned

Use /qbitfields to specify the sign of bitfields.

By default, bitfields are unsigned.

### /qdigraph

**Syntax:**

/qdigraph  
/qnodigraph

**Default:**

/qnodigraph

Use /qdigraph to permit digraph and keyword operators, and /qnodigraph to disallow them. See the *Language Reference* for details on digraph and keyword operators.

By default, digraph and keyword operators are not permitted.

## **/qlonglong Option •/Sd Option**

### **/qlonglong**

**Syntax:**

/qlonglong  
/qnoqlonglong

**Default:**

/qlonglong

Use /qlonglong to permit long long types, and /qnoqlonglong to disallow them in your program.

By default, /qlonglong is on.

### **/S**

**Syntax:**

/S<a|c|e|2>

**Default:**

/Se

Use /S to set the language level.

You can set the following language levels:

/Sa Conform to ANSI standards.



/Sc Allow constructs compatible with older levels of the C++ language.


**Note:** This option is valid only for C++ files.

/Se Allow all VisualAge for C++ language constructs. This is the default.



/S2 Conform to SAA Level 2 standards.

**Note:** This option is valid only for C files.

 See “Setting the Source Code Language Level” on page 71 for more information.

By default, the compiler allows all VisualAge for C++ language extensions.

### **/Sd**

**Syntax:**

/Sd[+|-]

**Default:**

/Sd-



Use /Sd to set the default file extension to .c. Any file without an extension is then assumed to be a C source file, and will be compiled and linked.

## /Sg Option

By default, you must specify the extension for a source file:

```
icc anthony.c
icc efrem.cpp
```

If you omit the extension, VisualAge for C++ compiler assumes that the file is an object file (.obj) and does not compile it, but only invokes the linker. The following commands are equivalent (assuming that /Sd+ has not been specified elsewhere, such as in the ICC environment variable):

```
icc dale
icc dale.obj
icc /Sd- dale
```

If you want the default file extension to be the default source file extension, use the /Sd+ option. For example, the following two commands are equivalent:

```
icc alistair.c
icc /Sd+ alistair
```

**Note:** The /Tc and /Tp options override the setting of /Sd. If you specify either /Tc or /Tp followed by a filename without an extension, the compiler looks for the name specified, **without an extension**, and treats the file as a C file if /Tc was specified or a C++ file if /Tp was specified. For example, given the following command:

```
icc /Tp xiaohu
```

the compiler searches for the file xiaohu and compiles it as a C++ file.

## /Sg

### Syntax:

```
/Sg[l] [,<r|*>]
/Sg-
```

### Default:

```
/Sg-
```



Use /Sg to set left and right margins of the input file and ignore text outside these margins. This option is useful when you use source files that contain columns of characters you want to ignore.

**Note:** This option is only valid for C files.

The left margin is set to *l*. The right margin can be the value *r*, or you can use an asterisk to denote no right margin. *l* and *r* must be from 1 to 65535, and *r* must be greater than or equal to *l*.

By default, no margins are set, and the compiler uses the entire input file.

## /Sh Option • /Si Option

### /Sh

**Syntax:**


/Sh[+|-]

**Default:**

/Sh-

Use /Sh to allow the use of data definition names (ddnames).

A ddname is the part of the data definition before the equal sign. Use a ddname in a call to fopen or freopen to refer to the data definition stored in the environment.

For more information on using ddnames,  see the *Programming Guide*.

By default, the compiler does not allow ddnames.

### /Si

**Syntax:**

/Si[+|-]  
/Si[dir][name]

**Default:**

/Si-

Use /Si to use precompiled header files, if they exist and are current.


If you specify a *name* or *dir* with the option, then the compiler looks for a precompiled header file with the name and in the directory you specify.

You can also use the **#pragma hdrfile** directive to tell the compiler what file to look for. You must still specify /Si.

If you do not specify a name or directory, the compiler looks for a file named csetc.pch (if the next source file is a C file) or csetcpp.pch (if the next source file is a C++ file), in the current working directory.

Use the /Fi option to create or update the precompiled header files. Use /Si and /Fi in combination to ensure that your precompiled header files are always up to date.

**Note:** The file you generate (/Fi) must be the same file you use (/Si). If you specify different filenames or directories with the two options, the name or directory specified last is used with both options. If you specify a filename or directory with **#pragma hdrfile**, it overrides the name or directory specified with the options.

 See “Using Precompiled Headers” on page 94 for more information.

By default, the compiler does not use precompiled header files.



## /Sm Option • /Sp Option

### /Sm

**Syntax:**  
/Sm[+|-]

**Default:**  
/Sm-

Use /Sm to ignore obsolete keywords, such as **near**, **far**, and **huge**.

By default, the compiler treats unsupported obsolete keywords like any other identifier.

### /Sn

**Syntax:**  
/Sn[+|-]

**Default:**  
/Sn-

Use /Sn to allow use of double-byte character set (DBCS). Compile with this option if your source files contain DBCS characters. Using /Sn increases your compile time.

By default, the compiler does not perform the checking needed to handle DBCS characters correctly.

### /Sp

**Syntax:**  
/Sp[1|2|4|8|16]

**Default:**  
/Sp8

Use /Sp to specify alignment or packing of data items within structures and unions.

By default, structures and unions are aligned along 8-byte boundaries (normal alignment).

/Sp is equivalent to /Sp1.

## /Sq Option • /Ss Option

### /Sq

**Syntax:**  
/Sq[*l*] [*r*,<*r*|\*>]  
/Sq-

**Default:**  
/Sq-



Use /Sq to specify columns in which sequence numbers appear, and ignore text in those columns. This option can be used when importing source files from other systems, and is provided primarily for compatibility with IBM C/370.

**Note:** This option is only valid for C files.

Sequence numbers appear between columns *l* and *r* of each line in the input source code. *l* and *r* must be from 1 to 65535, and *r* must be greater than or equal to *l*. If you do not want to specify a right column, use an asterisk for *r*.

By default, the compiler does not use sequence numbers.

### /Sr

**Syntax:**  
/Sr[+|-]

**Default:**  
/Sr-



Use /Sr to use old-style rules for type conversion. Old-style rules preserve the sign. They do not conform to ANSI standards.

By default, the compiler uses ANSI standard rules for type conversion. These rules preserve accuracy.

**Note:** This option is valid for C files only. C++ files must use the ANSI standard type conversion rules.

### /Ss

**Syntax:**  
/Ss[+|-]

**Default:**  
/Ss+



Use /Ss to allow the use of double slashes (//) for comments.

By default, the compiler allows double slashes to indicate comments in a C file.

**Note:** This option is only valid for C files. C++ allows double slashes to indicate comments as part of the language.

## /Su Option •/Tc Option

### /Su

**Syntax:**

/Su[+|-|1|2|4]

**Default:**

/Su-

Use /Su to control the size of enum variables. If you do not provide a size, all enum variables are made 4 bytes.

By default, the compiler uses the SAA rules: make all enum variables the size of the smallest integral type that can contain all variables.

You can specify the following sizes:

/Su[+] Make all enum variables 4 bytes.

/Su1 Make all enum variables 1 byte.

/Su2 Make all enum variables 2 bytes.

/Su4 Make all enum variables 4 bytes.

### /Sv

**Syntax:**

/Sv[+|-]

**Default:**

/Sv-

Use /Sv to allow the use of memory files.

For more information on using memory files,  see the *Programming Guide*.

By default, the compiler does not allow memory files.

### /Tc

**Syntax:**

/Tc *filename*

**Default:**

Compile \*.cpp and \*.cxx as C++,  
compile \*.c and unrecognized files as C.



Use /Tc to specify that the following file is a C file, regardless of its extension.

**Important:** The /Tc option **must** be immediately followed by a filename, and applies only to that file.

By default, the compiler compiles .cpp and .cxx files as C++ files, and .c and all other unrecognized files as C files.

## /Td Option •/Tp Option

### /Td

**Syntax:**

/Td[c|p]

**Default:**

/Td


Use /Td to specify a default language (C or C++) for files.

By default, the compiler compiles .cpp and .cxx files as C++ files, and .c and all other unrecognized files as C files.

You can change the default as follows:

/Tdc Compile all source and unrecognized files that follow on the command line as C files.

/Tdp Compile all source and unrecognized files that follow on the command line as C++ files.

 See “File Types” on page 64 for a list of file types the compiler recognizes.

**Note:** You can specify /Td anywhere on the command line to return to the default rules for the files that follow it.

### /Tp

**Syntax:**

/Tp *filename*

**Default:**

Compile \*.cpp and \*.cxx as C++,  
compile \*.c and unrecognized files as C.



Use /Tp to specify that the following file is a C++ file, regardless of its extension.

**Important:** The /Tp option **must** be immediately followed by a filename, and applies only to that file.

By default, the compiler compiles .cpp and .cxx files as C++ files, and .c and all other unrecognized files as C files.


---

## Preprocessor Options

Use these options to control the use of the preprocessor.

Note that the /Pc, /Pd, and /Pe options are actually suboptions of /P. Specifying /Pc- is the same as specifying /P+c- and causes only the preprocessor to be run.

### Using the Preprocessor

Preprocessor directives, such as **#include**, allow you to include C or C++ code from another source file into yours, to define macros, and to expand macros.  See the *Language Reference* for a list of preprocessor directives and information on how to use them.

If you run only the preprocessor, you can use the preprocessor output (which has all the preprocessor directives executed, but no code compiled) to debug your program. For example, all macros are expanded, and the code for all files included by **#include** directives appears in your program.

By default, comments in the source code are not included in the preprocessor output. To preserve the comments, use the /Pc option. For C programs, if you use // to begin your comments, you must also specify the /Ss option to include those comments in the preprocessor output.

The /P, /Pc, /Pd, and /Pe options can be used in combination with each other. For example, specify /Pcde to preserve comments, suppress #line directives, and redirect the preprocessor output to **stdout**.

### /D

#### Syntax:

/Dname  
/Dname[=n]

#### Default:

Define no macros on the command line.

Use /D to define preprocessor macro *name* to the value *n*. If /Dname is specified, *n* defaults to 1. If either /Dname= or /Dname:: is specified, the macro is set to a null string. Macros defined on the command line override macros defined in the source code.

If the value *n* is more than one word, delimit it with double quotes:

/Dname="a b c"

To define *n* to a string literal, delimit the string literal with /" at either end, and enclose the whole in double quotes:

## **/P Option •/Pd Option**

```
/Dname=" /"Some text/" "
```

Use the /U option to undefine macros on the command line.

By default, no macros are defined on the command line.

### **/P**

**Syntax:**

/P[+|-]

**Default:**

/P-

Use /P to run the preprocessor only, and create a preprocessor output file that has the same name as the source file, with the extension .i.

By default, both the preprocessor and the compiler run, and no preprocessor output is generated.

### **/Pc**

**Syntax:**

/Pc[+|-]

**Default:**

/P-

Use /Pc to run the preprocessor only, and create a preprocessor output file that includes the comments from the source code. The output file has the same name as the source file, with the extension .i.

Specify /Pc- to run the preprocessor only, and create a preprocessor output file with the comments stripped out. The output file has the same name as the source file, with the extension .i. /Pc- is equivalent to /P[+].

By default, both the compiler and preprocessor run, and no preprocessor output is generated.

### **/Pd**

**Syntax:**

/Pd[+|-]

**Default:**

/P-

Use /Pd to run the preprocessor only, and send the preprocessor output to **stdout**.

## /Pe Option • /U Option

Specify /Pd- to run the preprocessor only, and not redirect preprocessor output. Preprocessor output is written to a file that has the same name as the source file, with the extension .i. /Pd- is equivalent to /P[+].

By default, both the compiler and preprocessor run, and no preprocessor output is generated.

### /Pe

**Syntax:**

/Pe[+|-]

**Default:**

/P-

Use /Pe to run the preprocessor only, and suppress generation of #line directives in the preprocessor output. The output file has the same name as the source file, with the extension .i.

Specify /Pe- to run the preprocessor only, and generate #line directives in the preprocessor output. The output file has the same name as the source file, with the extension .i. /Pe- is equivalent to /P[+].

By default, both the compiler and preprocessor run, and no preprocessor output is generated.

### /U

**Syntax:**

/U<name>\*>

**Default:**

Retain macros.

Use /U to undefine macros.

Specify /Uname to undefine macro name.

Specify /U\* to undefine all macros.

**Note:** /U does not affect some macros, such as \_\_DATE\_\_, \_\_TIME\_\_, \_\_TIMESTAMP\_\_, \_\_FILE\_\_, and \_\_FUNCTION\_\_, nor does it undefine macros defined in source code.

Use the /D option to define or redefine macros on the command line. Macros defined on the command line override macros defined in the source code.

By default, the preprocessor retains all macros.


## /G Option

---

### Code Generation Options

Use these options to specify the type of code that the compiler will produce. The types of code include:

- Dynamically linked runtime libraries
- Statically linked runtime libraries
- Single-thread programs
- Multithread programs
- Subsystems

 See the *Programming Guide* for more information.

#### Notes:

1. The /Oi[+] option is more effective when /O[+] is also specified.
2. Using optimization (/O[+]) limits your use of the VisualAge for C++ Debugger to debug your code. The /Ti option is not recommended for use with optimization when you are debugging. You can still use the /Ti option for analysis with the Performance Analyzer.

## /G

#### Syntax:

/G<3|4|5>

#### Default:

/qtune=blend

Use /G to specify the type of processor your code will run on.

**Note:** The /qtune option provides more functionality than that offered by /G. Use /qtune instead.

By default, the compiler generates code targeted at all x86 processors. This is equivalent to /qtune=blend.

You can specify the following processors:

/G3    Optimize code for any 386-class processor.

      /qtune=386 is equivalent to /G3.

/G4    Optimize code for any 486-class processor.

      /qtune=486 is equivalent to /G4.

/G5    Optimize code for the Pentium processor.

      /qtune=pentium is equivalent to /G5.



## /Gd Option • /Ge Option


**Note:** To optimize code for the PentiumPro processor, you must specify `/qtune=pentiumpro`.

### /Gd

**Syntax:**  
`/Gd[+|-]`

**Default:**  
`/Gd-`

Use `/Gd` to dynamically link to the runtime library. Your `.exe` or `.dll` file will call functions from VisualAge for C++ DLLs, and must have access to these DLLs to run.

 See “Static and Dynamic Linking” on page 92 for more information.

By default, the runtime library is statically linked. VisualAge for C++ functions are copied into your `.exe` or `.dll` file from the VisualAge for C++ `.lib` files, and do not need access to the VisualAge for C++ DLLs.

### /Ge

**Syntax:**  
`/Ge[+|-]`

**Default:**  
`/Ge[+]`

Use `/Ge-` to build a `.dll` file.

By default (`/Ge[+]`), the compiler builds an `.exe` file.

The VisualAge for C++ libraries provide two initialization routines, one for executable modules and one for DLLs. For each object file, the compiler must include a reference to the appropriate initialization routine. The name of this routine is then passed to the linker when the file is linked. Use the `/Ge` option at compile time to tell the compiler which routine to reference.

The `/Ge-` option causes the compiler to generate a reference to `_dllentry` for every module compiled. The `/Ge+` option generates a reference to `_exeentry` only if a `main` function is found in the source, or generates a reference to `_winentry` only if a `WinMain` function is found in the source. If no `main` or `WinMain` function is included, no linking reference is generated.



If you want to create a library of objects that can be linked into either an executable file or a DLL, use the `/Ge+` option when you compile. Typically, none of these objects would contain a definition of `main`.

## /Gf Option •/Gh Option

If one of the objects **does** contain a definition of `main`, you can override the `/Ge` option when you link your files, as follows:

1. Create a source file that defines `_exeentry`
2. Compile it into an `.obj` file, with the options `/C+` (create `.obj` file only, no linking) and `/Ge-` (compile for DLL)
3. Link the resulting object file with your other object files

When you link, the definition of `_exeentry` resolves any references in your other object files. Because you compiled the file with `/Ge-`, it contains a reference to `_dllentry`, and the linker links in the correct initialization routines.

### /Gf

**Syntax:**

`/Gf[+|-]`

**Default:**

`/Gf+`

Use `/Gf` to specify fast floating-point execution.

If your program does not need to abide by ANSI rules regarding the processing of double and float types, you can use this option to increase your program's performance. Because the fast floating-point method does not perform all the conversions specified by the ANSI standards, the results obtained may differ from results obtained using ANSI methods, but are often more precise.

By default, the compiler uses fast floating-point execution.

### /Gh


**Syntax:**

`/Gh[+|-]`

**Default:**

`/Gh-`

Use `/Gh` to enable code to be run by the Performance Analyzer and other profiling tools by generated profiler hooks in function prologs.

For more information on the Performance Analyzer,  see Part 7, “Performance Execution Trace Analyzer” on page 429.

**Note:** To enable code for the Performance Analyzer, you must also specify `/Ti`.

By default, code is not enabled for the Performance Analyzer.

## /Gi Option •/G1 Option

### /Gi

**Syntax:**  
/Gi [+|-]

**Default:**  
/Gi-

Use /Gi to specify fast integer execution.

If you are shifting bits by a variable amount, you can use fast integer execution to ensure that for values greater than 31, the bits are shifted by the result of a modulo 32 of the value. Otherwise, the result of the shift is 0.

**Note:** If your shift value is a constant greater than 32, the result will always be 0.

By default, the compiler does not use fast integer execution.


### /G1

**Syntax:**  
/G1 [+|-]

**Default:**  
/G1-

Use /G1 to remove unreachable functions. The compiler prepares the object code for optimization by the linker, and then passes the /OPTFUNC option to the linker. The linker removes functions that are:

- Not referenced anywhere in the object code
- Rendered unreferenced by the removal of other functions
- Not exported for use in other files

 See “EXPORTS” on page 537 for more information on exporting functions.

When the function is removed, any additional functions that were required only by that function are also removed. Removing the functions and code reduces the size of your .exe or .dll output file.

If you are compiling and linking in separate steps, you must compile with /G1 and then link with /OPTFUNC to perform this optimization.

By default, the linker does not remove unreachable functions.

**Performance Consideration:** Optimized linking generally takes longer than regular linking, because of the extra processing that the linker performs. However, if the optimization is effective enough, it can actually speed up the linking process, because there is less information to write to file. Generally, you may want to compile without the /G1 option, until your code is tested and stable.

## **/Gm Option • /Gs Option**

### **/Gm**


**Syntax:**

/Gm[+|-]

**Default:**

/Gm-

Use /Gm to link with the multithread version of the library.

 See “Choosing Your Runtime Libraries” on page 91 and “Using the Multithread Library” on page 93 for more information.

By default, object files are linked with the single-thread version of the library.

### **/Gn**

**Syntax:**

/Gn[+|-]

**Default:**

/Gn-

Use /Gn to suppress linker information about the default libraries defined by the /Gd, /Gm, /Ge, and /Rn options. All libraries must then be explicitly identified at link time.

By default, the compiler embeds the names of the default libraries in the object files, for use by the linker.

### **/Gs**

**Syntax:**

/Gs[+|-]

**Default:**

/Gs-

Use /Gs to remove stack probes from the generated code.

For more information on stack probes,  see “Controlling Stack Allocation and Stack Probes” on page 102.

By default, stack probes are not removed.

## /Gu Option •/Gw Option

### /Gu


**Syntax:**

/Gu[+|-]

**Default:**

/Gu-

Use /Gu to tell the intermediate linker that data defined in the intermediate link is not used by external functions. The data is used only within the intermediate files being linked, with the exception of data that is exported using `_Export`, **#pragma export**, or a .DEF file.

When you specify /Gu, the intermediate code linker can optimize code more effectively.  See “Using the Intermediate Code Linker” on page 81 for more information about the intermediate code linker.

By default, external functions may use data defined in the intermediate files being linked.

### /Gw

**Syntax:**

/Gw[+|-]

**Default:**

/Gw-

Use /Gw to generate an FWAIT instruction after each floating-point load instruction. This allows the program to take a floating-point stack overflow exception immediately after the load instruction that caused it.

**Note:** This option is not recommended because it increases the size of your executable file and greatly decreases its performance. You do not need this option unless you call assembler code that leaves a different number of values on the floating-point stack.

By default, FWAIT instructions are not generated after each floating-point load instruction.

## /Gx Option •/M Option

### /Gx

**Syntax:**


/Gx[+|-]

**Default:**

/Gx-



Use /Gx to remove C++ exception-handling information.

For more information on C++ exception handling,  see the *Programming Guide* and the *Language Reference*.

By default, C++ exception-handling information is not removed.

**Note:** This option is valid for C++ files only.

### /M

**Syntax:**

/M<p|s|c|t>

**Default:**

/Mp

Use /M to set the calling convention, as follows:

**Option Calling Convention**

/Ms     **\_System** calling convention

/Mc     **\_\_cdecl** calling convention

/Mt     **\_\_stdcall** calling convention

/Mp     **\_Optlink** calling convention

The default is the **\_Optlink** calling convention (/Mp).

You must include the header files for libraries that use a different calling convention from the one you specify. The libraries use the following calling conventions:

**VisualAge for C++ libraries**


Functions use the **\_Optlink** calling convention. Include the VisualAge for C++ library header files to call VisualAge for C++ functions when you set /Ms, /Mc, or /Mt.

**Toolkit libraries**

APIs use the **\_\_stdcall** calling convention. Include the Toolkit library header files to call Windows APIs when you set /Mp (the default), /Mc, or /Mt.

If you do not include the header files, then your code will attempt to call functions with the calling convention you set, rather than with the calling convention the function requires.

## /Nd Option •/Nt Option

 See “Setting the Calling Convention” on page 90 for more information on calling conventions.

### /Nd

**Syntax:**

/Ndname

**Default:**

Use DATA32, BSS32, and  
CONST32\_RO

Use /Nd to specify the names of default data, uninitialized data, and constant sections as *nameDATA32*, *nameBSS32*, and *nameCONST32\_RO*, respectively. You can then give the sections special attributes, with the linker option /SECTION. The renamed sections are not placed in the default data group (DGROUP).

**Notes:**

1. While the /Nd option allows you to rename CONST32\_RO and BSS32, you do not need to rename them in order to assign them special attributes.
2. CONST32\_RO is READONLY and SHARED by default.

If you do not use the /Nd option, the default names are DATA32, BSS32, and CONST32\_RO.

### /Nt

**Syntax:**

/Ntname

**Default:**

Use CODE32

Use /Nt to specify the name of default code or text sections as *nameCODE32*. You can then give the sections special attributes, with the linker option /SECTION.

If you do not use the /Nt option, the default name is CODE32.

## /Nx Option • /O Option

### /Nx

**Syntax:**

/Nxname

**Default:**

EH\_CODE and EH\_DATA



Use /Nx to specify the names of one code and one data section that contain information relating to C++ exception handling. This information is only used during the processing of a throw statement. Separating the sections from the default code and data segments improves the paging behavior of your program.

The code section is named *name\_CODE* of class CODE. The data section is named *name\_DATA* of class DATA.

**Note:** This option is valid for C++ files only.

By default, the sections are named EH\_CODE and EH\_DATA.

### /O

**Syntax:**


/O[+|-] | 2 | 3

**Default:**

/O-

Use /O to optimize code for speed.

/O2 and /O3 provide the same level of optimization as /O.

**Note:** Do not optimize code if you want to use debugging or diagnostic options. The debugger may operate unpredictably with optimized code.  See “Generating Debugger Information” on page 77 for more information.

When you specify /O, the following optimization options are turned on by default:

/Oi Turn on inlining.

/Os Invoke the instruction scheduler.

By default, the code is not optimized.

See the optimization section in the *Programming Guide* for more details.



## /Oc Option • /Oi Option

### /Oc

**Syntax:**  
/Oc[+|-]


**Default:**  
/Oc-

Use /Oc to optimize code for size as well as speed. You must also specify /O.



/Oc performs the same set of optimizations as /O, except for those that increase the size of the code. Code optimized with /Oc is **not** slower than unoptimized code, and is likely to be faster, though not as fast as code optimized with /O on its own.

For example, /Oc stops loops from being unrolled, and stops most inlining (unless you specified a conflicting /Oi value).

**Note:** Do not optimize code if you want to use debugging or diagnostic options. The debugger may operate unpredictably with optimized code.  See “Generating Debugger Information” on page 77 for more information.

By default, the code is not optimized.

### /Oi

**Syntax:**  
/Oi[+|-]  
/Oi*value*

**Default:**  
/Oi-, /Oi+ when /O+  
/Oi0

Use /Oi to control inlining of user code.


By default, the compiler does not inline user code, unless you specify /O[+]. When you specify /O[+], /Oi[+] becomes the default.

You can specify the following types of inlining:

/Oi+      Inline all user functions that are qualified with the **\_Inline** or **inline** keyword.

/Oi-      Do not inline any user code.

/Oi*value*      Inline all user functions qualified with the **\_Inline** or **inline** keyword or that are smaller than *value* in abstract code units.

 See “Inlining User Code” on page 85 for more information.

## /Ol •/Op Option

### /Ol

**Syntax:**

/Ol [+|-]

**Default:**

/Ol-

Use /Ol to pass code through the intermediate linker before generating an object file.

 See “Using the Intermediate Code Linker” on page 81 for more information.

By default, code is not passed through the intermediate linker.

### /Om

**Syntax:**

/Om [+|-]

**Default:**

/Om-

Use /Om to limit the working set size for the compiler to approximately 35M.

The compiler may use a large amount of memory when inlining user code, especially when performing automatic inlining at large thresholds (/Oi50 and higher).

By default, there is no limit to the compiler's working set size.

**Note:** Because /Om[+] can cause the compiler to disregard some inlining opportunities, code generated with /Om- (the default) may be more efficient.

### /Op

**Syntax:**

/Op [+|-]

**Default:**

/Op+

Use /Op to perform optimizations involving the stack pointer. To use this option, you must also specify /O[+].

By default, optimizations involving the stack pointer are always performed.

Specify /Op- when you optimize code that directly manipulates the stack pointer. Using /Op- decreases the performance of your .exe or .dll file.

## /Os Option • /qisolated\_call Option

### /Os

**Syntax:**

/Os[+|-]

**Default:**

/Os-  
/Os+ when /O+

Use /Os to invoke the instruction scheduler. To use this option, you must also specify /O[+].

By default, the instruction scheduler is invoked when you have specified /O[+]. Otherwise, the instruction scheduler cannot be invoked.

### /qalias

**Syntax:**

/qalias=*option*[:*option*]...

**Default:**

None.

Use /qalias to specify what kind of aliasing can take place in the compilation unit. Typically, less aliasing often allows the compiler to perform more aggressive optimizations.

*option* can be:



**TYPEptr** Pointers to different types are not aliased.

**Note:** This suboption is valid only for C files.



**ALLPtrs** Pointers are not aliased.

**Note:** This suboption is valid only for C files.



**ADDRtaken** Variables are disjoint from pointers unless address are taken.

**Note:** This suboption is valid only for C files.

**ANSI** Type-based aliasing is used during optimization.

### /qisolated\_call

**Syntax:**

/qisolated\_call=*fname*[:*fname*]...

**Default:**

None.

Use /qisolated\_call to indicate that invocation of the specified functions has no side effects; that is, entities in the invoking program are not affected by invocations of the specified functions. This option provides opportunities for optimization.

## **/qtune Option •/qwin32s Option**

*fname* is any function name.

By default, no assertions are made about side effects.

### **/qtune**

**Syntax:**  
*/qtune=option*

**Default:**  
*/qtune=blend*

Use */qtune* to specify the type of processor for which the executable program is optimized.

*option* is one of the following:

<b>386</b>	Produce object code optimized for 386-class processors
<b>486</b>	Produce object code optimized for 486-class processors
<b>blend</b>	Produce object code optimized for all x86 processors
<b>pentium</b>	Produce object code optimized for Pentium processors
<b>pentiumpro</b>	Produce object code optimized for PentiumPro processors

The code can run on any x86 processor, but is specifically tuned for the target processor. The compiler also performs optimizations that help other processors if those optimizations do not in turn detract from the performance of the target processor. For example, code optimized for a 486 processor also includes any Pentium optimizations that do not diminish 486 execution.

If you do not know what processor your application will be run on, specify */qtune=blend*. The compiler includes optimizations that help all x86 processors. It also provides optimizations that specifically help execution on Pentium and 486 processors, if those optimizations do not significantly detract from the performance on other x86 processors.

By default, */qtune=blend* is specified.

### **/qwin32s**

**Syntax:**  
*/qwin32s*  
*/qnowin32s*

**Default:**  
*/qnowin32s*

Use */qwin32s* to help port your program to the Win32s runtime environment.

## **/R Option**

Although a program compiled with /qwin32s can run under Windows NT and Windows 95, performance may be slower than if the program had not been compiled with the option.

The default is /qnowin32s.

See the *Programming Guide* for details on creating programs to run in the Win32s runtime environment.

### **/R**

**Syntax:**


/R[e|n]

**Default:**

/Re

Use /R to control the executable runtime environment.

Specify /Rn to generate executable code that can be used as a subsystem without a runtime environment.

For more information on developing subsystems,  see “Enabling Subsystem Development” on page 94.

By default (/Re), the compiler generates executable code that runs in a VisualAge for C++ environment.

## SOM Options

---

### System Object Model (SOM) Options

This section describes the compiler options available for SOM support in VisualAge for C++. See the *Programming Guide* for background information on the reasons for these options and on their uses. The following options are described:

“/Fr” on page 170

“/Fs” on page 170

“/Ga”

“/Gb” on page 169

“/Gz” on page 169

“/Xs” on page 169

SOM options that affect the same settings as SOM pragmas are effective except when overridden by those pragmas. For example, the /Ga compiler option, which causes all classes to implicitly derive from SOMObject, turns the **SOMAsDefault** pragma on at the start of the translation unit. This pragma remains in effect until a **#pragma SOMAsDefault(off|pop)** is encountered in the translation unit. See the *Programming Guide* for more information on the relationship between SOM pragma settings and SOM options.

In addition to the compiler options, the compiler defines a macro, **\_\_SOM\_ENABLED\_\_**, whose value corresponds to the level of SOM support provided by the compiler. If SOM support is not provided for a particular release of the compiler, **\_\_SOM\_ENABLED\_\_** is not predefined.

The macro's value is a positive integer constant. For the first SOM-supporting release of VisualAge for C++, the level of SOM supported is SOM 2.1, so the macro has the value 210.

#### /Ga

**Syntax:**

/Ga[+|-]

**Default:**

/Ga-

This option turns on implicit SOM mode, and also causes the file `som.h` to be included. It is equivalent to placing **#pragma SOMAsDefault(on)** at the start of the translation unit.

## SOM Options

All classes are implicitly derived from `SOMObject` until a **#pragma SOMAsDefault(off)** is encountered.

For further details, see the *Programming Guide*.

### /Gb

**Syntax:**  
`/Gb[+|-]`

**Default:**  
`/Gb-`

This option instructs the compiler to disable direct access to attributes. Instead, the `get` and `set` methods are used. This is equivalent to specifying **#pragma SOMNoDataDirect(on)** as the first line of the translation unit.

For further details, see the *Programming Guide*.

### /Gz

**Syntax:**  
`/Gz[+|-]`

**Default:**  
`/Gz-`

Use this option to initialize SOM classes at their point of first use during the execution of your program.

By default, all SOM classes statically used in your program are initialized at static initialization time. This makes your program smaller, but may result in the initialization of classes that are not dynamically used.

With any setting of this option, any reference to a static member of a SOM class will cause the class to be initialized.

For further details, see the *Programming Guide*.

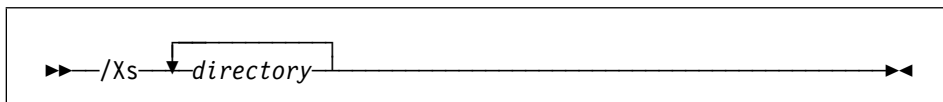
### /Xs

**Syntax:**  
`/Xs<directory|->`

**Default:**  
`/Xs-`

Use this option to exclude files in the specified directories when implicit SOM mode is turned on (when classes are implicitly derived from SOM). The syntax of this option is:

## SOM Options



where *directory* is the name of the directory or directories you want to exclude. Directory names are separated with a semicolon (;).

This option is useful for mixing implicit SOM mode with existing include files that include declarations of classes you do not want to be implicit SOM classes.

### /Fr

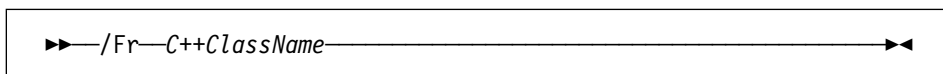
**Syntax:**

`/Fr<classname>`

**Default:**

None

Use this option to have the compiler write the release order of the specified class to standard output. The release order is written in the form of a **SOMReleaseOrder** pragma. You can capture the output from this option when developing new SOM classes, and include the pragma in the class definition. The syntax of the option is:



For further details, see the *Programming Guide*.

### /Fs

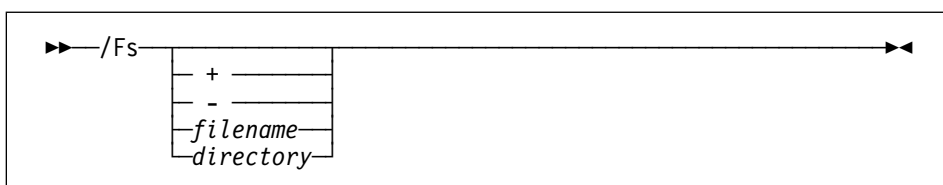
**Syntax:**

`/Fs[+|-|file| directory]`

**Default:**

`/Fs-`

Use this option to have the compiler generate an IDL file if a file with an `.hh` extension is explicitly specified on the command line. The syntax of the option is:



where:



## **/? Option**

`/Fs<+>` specifies that an IDL file will be created for every `.hh` file that is specified on the command line and is in the current directory. This is the default.

`/Fs filename.ext` is like `/Fs +`, but the IDL file that is created will have the specified filename. If you do not specify an ext, the extension will be `idl`.

`/Fs directory_name` is like `/Fs +`, but the IDL file that is created will be put in the directory `directory_name` rather than the current directory. `directory_name` must end with a backslash `"\"`.

`/Fs-` specifies that no IDL file should be created.

---

## **Other Options**

Use these options to control linker parameters, logo display, default char type, and other VisualAge for C++ settings.

### **Examples**



- Passing a parameter to the linker:

```
icc /B"/NOI" fred.c
```

The `/NOI` option tells the linker to preserve the case of external names in `fred.obj`.

- Imbedding a version string or copyright:

```
icc /V"Version 1.0" fred.c
```

This imbeds the version notice in `fred.obj`.

## **/?**

### **Syntax:**

`/?`

### **Default:**

None


Use `/?` to display a list of compiler options with descriptions.


## **/B Option •/H Option**

### **/B**

**Syntax:**  
`/B"options"`

**Default:**  
`/B""`

Use `/B` to pass the *options* string to the linker as parameters. The `icc` default parameters are also passed.  See “Summary of Linker Options” on page 201 for information about the options you can pass to the VisualAge for C++ linker.


By default, only the `icc` default parameters are passed to the linker.  See “Linking through the Compiler” on page 182 for a description of the options passed to the linker by default.

### **/C**

**Syntax:**  
`/C[+|-]`

**Default:**  
`/C-`

Use `/C` to perform a compile only, without linking.

When you link separately, you need to specify options that the compiler normally passes to the linker.  See “Linking through the Compiler” on page 182 for a description of these options.

By default, code is both compiled and linked.

### **/H**

**Syntax:**  
`/Hnum`

**Default:**  
`/H255`

Use `/H` to set the significant length of external names. The first *num* characters of an external name are set as significant. The value of *num* must be between 7 and 255 inclusive.

By default, the first 255 characters of external names are significant.

## /Q Option • /qautothread Option

### /Q

**Syntax:**

/Q[+|-]

**Default:**

/Q-

Use /Q to stop the compiler logo from appearing when you invoke the compiler.

By default, the compiler logo appears on **stderr**.

### /qautoimported

**Syntax:**

/qautoimported  
/qnoautoimported

**Default:**

/qnoautoimported

Use /qautoimported to import all external references. However, note that more efficient code is generated if, in place of /qautoimported, imports are explicitly indicated using the **\_Import** keyword or the **\_declspec(dllimport)** qualifier.

By default, external references are not imported.

### /qautothread

**Syntax:**

/qautothread  
/qnoautothread

**Default:**

/qnoautothread

Use /qautothread to turn all static, nonconstant data into thread-local data by adding the **\_\_thread** keyword to the data declarations. This may be required in order to run the program in the WIN32s runtime environment.

For some data declarations that are not altered by /qautothread, you may have to add the **\_\_thread** keyword manually.

By default, the **\_\_thread** keyword is not automatically added to variable declarations.

See the *Programming Guide* for details on creating programs to run in the WIN32s runtime environment.

## **/qignprag Option •/qmakedep Option**

### **/qignprag**

**Syntax:**

/qignprag=<disjoint|isolated|all>

**Default:**

None.

Use /qignprag to either ignore references to #pragma disjoint, #pragma isolated\_call, or both.

By default, these pragmas are not ignored.

### **/qlibansi**

**Syntax:**

/qlibansi  
/qnolibansi

**Default:**

/qnolibansi

Use /qlibansi to instruct the compiler that all functions with the name of an ANSI C library function are the system functions, not user-defined functions.

With /qlibansi, the optimizer may generate code that performs better because it already knows the behavior of a system function (for example, whether or not it causes side effects).

By default, the compiler does not assume that functions with the same name as ANSI C library functions are system functions.

### **/qmakedep**

**Syntax:**

/qmakedep

**Default:**

None.

Use /qmakedep to generate make file information and output it to a dependencies (.u) file. This information can then be used by the NMAKE utility.

An example of the output format is as follows:

```
myprog.obj: myprog.c
myprog.obj: C:\ibmcppw\include\stdio.h
myprog.obj: C:\ibmcppw\include\stdlib.h
```

By default, this information is not generated.

## /qro Option • /V Option

### /qro

**Syntax:**  
/qro  
/qnoro

**Default:**  
/qro

Use /qro to specify that string literals be placed in read-only storage. If /qnoro is specified, string literals are placed in read/write storage.

By default, string literals are placed in read-only storage.

### /qsomvolattr

**Syntax:**  
/qsomvolattr  
/qnosomvolattr

**Default:**  
/qnosomvolattr



Use /qsomvolattr to specify that attribute prototypes are volatile.

By default, attribute prototypes are not declared volatile.

### /V

**Syntax:**  
/V"*string*"

**Default:**  
No string.

Use /V to include a version string in the object and executable output files. The version string is set to *string*. The length of the string can be from 1 to 256 characters.

By default, no version string is set.

## /qrtti Option

---

### Options for New ANSI Standards

These options support new ANSI standards.

#### /qrtti

**Syntax:**  
*/qrtti=rtoption*  
*/qnortti*

**Default:**  
*/qnortti*



Use */qrtti* to generate information for the typeid operator and the dynamic\_cast operator.

The suboptions (*rtoption*) are:

- |                    |                                                                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ALL</b>         | The compiler generates the information needed for the RTTI typeid and dynamic_cast operators.                                                      |
| <b>TYPEinfo</b>    | The compiler generates the information needed for the RTTI typeid operator, but the information needed for dynamic_cast operator is not generated. |
| <b>DYNAmiccast</b> | The compiler generates the information needed for the RTTI dynamic_cast operator, but the information needed for typeid operator is not generated. |

---

## Part 3. Linking Your Program

This part of the *User's Guide* describes the VisualAge for C++ linker, which links the object files produced by the compiler into executable files: .exe files or .dll files. By default, the compiler invokes the linker for you.

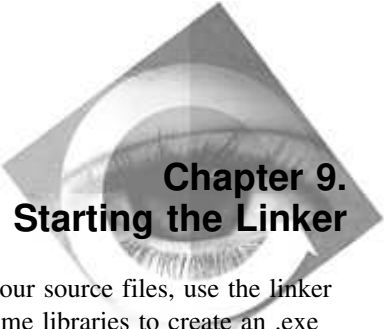
---

<b>Chapter 9. Starting the Linker</b>	179
Linking within WorkFrame	179
Linking from the Command Line	180
Linking through the Compiler	182
Linking from a Make File	183
 <b>Chapter 10. Optimized Linking</b>	 185
 <b>Chapter 11. Input and Output</b>	 187
Specifying Object Files	189
Specifying Executable Output Type	190
Generating a Map File	191
Linker Return Codes	192
 <b>Chapter 12. Linking with Library Files</b>	 193
Linking with .lib Files	194
Linking to Dynamic Link Libraries	194
 <b>Chapter 13. Setting Linker Options</b>	 197
Specifying Numeric Arguments	199
Summary of Linker Options	201
Linker Options	202

---







## Chapter 9. Starting the Linker

Once the compiler has created object modules out of your source files, use the linker to link them together with the VisualAge for C++ runtime libraries to create an .exe file or .dll file. By default, the `icc` command invokes the linker for you.

There are several ways you can start the linker:

- From the popup menu of an object file in a WorkFrame project
- From the command line
- Through the compiler, which automatically invokes the linker
- Through a makefile, which invokes both the compiler and the linker

---

### Linking within WorkFrame

To use the linker through WorkFrame, do the following:

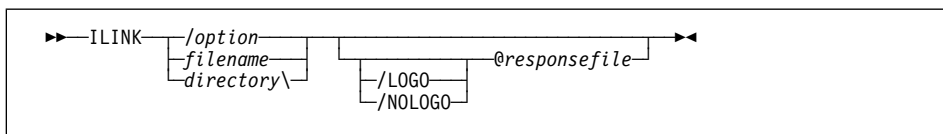
1. Double click on the WorkFrame icon to open WorkFrame.
2. Use the initial WorkFrame dialog to either open an existing project or create a new one. These actions are also choices on the WorkFrame **Project** pull-down menu. Once a project has been opened or created, its files are listed in the WorkFrame window.
3. Customize the linker options from the **Options** pull-down menu, if you do not want to use the defaults. The **Options** menu contains choices that allow you to specify options for other actions (e.g., compile). Use Build Smarts to easily set build options, such as debug, that affect more than one action. You may also want to customize the project settings by selecting **Settings** on the **View** pulldown menu.
4. Select **Build** from either the **Project** pull-down menu or the project toolbar. Your project is built using the linker as required.

## Starting the Linker

---

### Linking from the Command Line

Specify the `ilink` command followed by any sequence of options, file names, or directories, separated by space or tab characters.



The linker recognizes the input as follows:

<b>Options</b>	Start with a / or - character.
<b>Directories</b>	End with a / or \ character.
<b>Response files</b>	Start with the @ character.
<b>Export files</b>	End with the .exp extension.
<b>Library files</b>	End with one of these extensions: <ul style="list-style-type: none"><li>• .a</li><li>• .imp</li><li>• .lib</li></ul>
<b>Resource files</b>	End with the .res extension.
<b>Object files</b>	Any other input. You must enter at least one object file.

You can specify the name of the output file with the `/OUT` option. You can specify the name of a map file with the `/MAP` option.

In addition to the libraries you specify, by default the linker searches the VisualAge for C++ runtime libraries defined in your object files at compile time and the system import library, `kernel32.lib`. See “Choosing Your Runtime Libraries” on page 91 for more information on setting the default libraries.

The directories you specify become part of the linker's search path, before any directories set in the `LIB` environment variable. See “Search Rules” on page 187 and “Specifying Directories” on page 188 for more information.

You can also use wildcard characters to specify multiple object files. For example, use `*.obj` to specify all the object files in a directory.

**Note:** If the linker cannot find a file, it stops linking. The linker does not assume the `.obj` extension for a file: if you specify a file with no extension, then the linker looks for that file with no extension.

## Starting the Linker

### Examples

The following command links the object files `fun.obj`, `text.obj`, `table.obj`, and `care.obj`. The linker searches for unresolved external references in the library file `xlib.lib` and in the default libraries. Since there is no name provided for the executable file, it is named `fun.exe`, taking the filename of the first object file and the default extension `.exe`. The linker also produces a map file, `funlist.map`.



```
ILINK /MAP:funlist fun.obj text.obj table.obj care.obj xlib.lib
```

The following command links the files `main.obj`, `getdata.obj`, and `printit.obj` into an executable file named `main.exe`, and produces a map file named `main.map`.

```
ILINK /MAP main.obj getdata.obj printit.obj
```

The following command links `getdata.obj` and `printit.obj` into a DLL named `getdata.dll`. The export file, `getdata.exp`, specifies the functions that are exported from `getdata.dll`. See Part 8, “Managing Libraries” on page 517 for details on creating an export file.

```
ILINK getdata.obj printit.obj /OUT:getdata.dll /DLL getdata.exp
```

### Using Response Files

Instead of specifying linker input on the command line, you can put options and filename parameters in a response file. You can combine the response file with options and parameters on the command line.

When you invoke the linker, use the following syntax:

```
ILINK @responsefile
```

where *responsefile* is the name of the response file. The @ symbol indicates that the file is a response file. If the file is not in the working directory, specify the path for the file as well as the file name. If the linker cannot find a file, it stops linking.



You can begin using a response file at any point on the linker command line. Although multiple response files can be specified on the command line, they cannot be nested.

Options can appear anywhere in the response file. If an option is not valid, the linker generates an error message and stops linking.

## Starting the Linker

### Example

The response file named `fun.lnk` contains the following:



```
/DEBUG /MAP  
fun.obj text.obj table.obj care.obj  
/exec  
/map:funlist  
graf.lib
```

When you enter:


```
ILINK @fun.lnk
```

the linker does the following:

- Links the four object modules `fun.obj`, `text.obj`, `table.obj`, and `care.obj` into an `.EXE` file named `fun.exe`. Because no output type is specified, the linker defaults to `.exe`.
- Generates the map file `funlist.map` (assuming the extension `.map`).
- Preserves debugging information (because of the `/DEBUG` option).
- Links any needed routines from the library file `graf.lib`, and from the default VisualAge for C++ libraries specified in the object files.

---


## Linking through the Compiler

When you invoke the VisualAge for C++ compiler, it compiles the object files from your source code and then automatically starts the linker, to link the object files into an `.exe` or `.dll` file. Use the `/B` compiler option to pass options to the linker. The compiler does not pass any default parameters to the linker.  See “Linker Options” on page 202 for more information on these linker options.

If you do not want the compiler to start the linker, specify the `/C` compiler option. You can then invoke the linker in a separate step.

---

### Linking from a Make File

Use a makefile to organize the sequence of actions (such as compiling and linking) required to build your project. You can then invoke all the actions in one step. The NMAKE utility saves you time by performing actions on only the files that have changed, and on the files that incorporate or depend on the changed files.  See Chapter 63, “Program Maintenance Utility (NMAKE)” on page 709 for more information.

You can write the makefile yourself, or you can use WorkFrame to manage the makefile. When you build through WorkFrame, a makefile is created and maintained automatically.

## Starting the Linker



## Chapter 10. Optimized Linking


### Removing Unreachable Functions

Just as the compiler can optimize your source code by removing or replacing instructions, the linker can optimize your object code, including code in libraries you are linking in, by removing unreferenced functions. When the function is removed, any code that was required only by that function is also removed, including any other functions that were referenced only by that function. This reduces the size of your output file.



Compile with the option `/G1`, and then link with the option `/OPTFUNC`, to remove functions that are:

- Not referenced in any input file
- Rendered unreferenced by the removal of other functions
- Not exported for use in other files

 See “EXPORTS” on page 537 for more information on exporting functions and data.


If you are compiling and linking in one step, you do not need to specify `/OPTFUNC`. The compiler passes `/OPTFUNC` to the linker automatically when you specify `/G1`.

### Performance Consideration

Optimized linking generally takes longer than regular linking, because of the extra processing that the linker performs. However, if the optimization is effective enough, it can actually speed up the linking process, because there is less information to write to file. Generally, you may want to link without the `/OPTFUNC` option until your code is tested and stable.

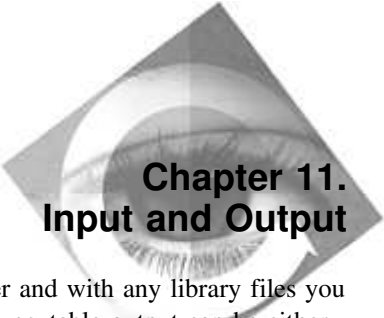
The linker produces slightly faster and more compact code and files by grouping neighboring sections that have similar attributes.

Specify `/DBGPACK` when you are debugging, to reduce the size of the executable file and potentially improve debugger performance.

 See “Linker Options” on page 202 for more information on these and other linker options.

## **Optimized Linking**





## Chapter 11. Input and Output

The linker takes object files, links them with each other and with any library files you specify, and produces an executable output file. The executable output can be either an executable program file (extension `.exe`) or a dynamic link library (extension `.dll`).

The linker optionally produces a map file, which provides information about the contents of the executable output.

### Input

*options*

*object files (\*.obj)*

*static library files (\*.lib)*

*import libraries (\*.lib)*

*export files (\*.exp)*

*resource files (\*.res)*

### Output

*executable file (.exe, .dll)*

*map file (.map)*


*return code*

## Search Rules

When searching for an object (`.obj`) or library (`.lib`), the linker looks only in the specified directory if a path is indicated. If a path is not specified, the linker looks in the following locations in this order:

1. The current directory. Default libraries do not include path specifications.

**Note:** If you specify a path with the file, the linker searches only that path, and stops linking if the file cannot be found there.

2. Any directories entered by themselves on the command line (they must end with a slash (/) or backslash (\) character).  See “Specifying Directories” on page 188 for more information.
3. Any directories listed in the LIB environment variable.

If the linker cannot locate a file, it generates an error message and stops linking.

## Linker Input and Output

### Example



A response file contains the following information:

```
FUN.OBJ TEXT.OBJ TABLE.OBJ CARE.OBJ  
NEWLIBV3.LIB  
C:\TESTLIB\
```

The linker links four object files to create an executable file named FUN.EXE. The linker searches NEWLIBV3.LIB before searching the default libraries to resolve references.


To locate NEWLIBV3.LIB and the default libraries, the linker searches the following locations in this order:

1. The current directory (because NEWLIBV3.LIB was entered without a path)
2. The C:\TESTLIB\ directory
3. The directories listed in the LIB environment variable

### Specifying Directories



To have the linker search additional directories for input files, specify a drive or directory by itself on the command line. Specify the drive or directory with a slash (/) or backslash (\) character at the end so the linker will recognize it as a path.

The linker will search the the paths you specify before it searches the paths in the LIB environment variable.  See “Search Rules” on page 187 for more information.

## Linker Input and Output


### Filename Defaults

If you do not enter a file name, the linker assumes the defaults shown below.

*Figure 25. Linker Filename Defaults*

File	Default Filename
Object files	None. You must enter at least one object file name.
Output file	The base name of the first object file.
Map file	The base name of the output file.
Library files	The default libraries defined in the object files. Use compiler options to define the default libraries. See “Choosing Your Runtime Libraries” on page 91 for more information. Any additional libraries you specify are searched before the default libraries.

### Specifying Object Files

When you invoke the linker from the command line, the linker assumes that any input it cannot recognize as other files, options, or directories must be an object file. Use a space or tab character to separate files.  See “Linking from the Command Line” on page 180 for more information on how the linker interprets input.

You can also use wildcard characters to specify multiple object files. For example, use \*.obj to specify all the object files in a directory.

When you invoke the linker through the compiler, the compiler automatically passes the object files it creates to the linker, as well as passing any object files you specify on the compiler command line or in the ICC environment variable. (You cannot specify object files through the ILINK environment variable.)

The linker accepts object files compiled or assembled:

- In 32-bit OMF format
- For Windows NT version 3.51 (or higher) or Windows 95
- For the 80386, 80486, and Pentium microprocessors

You must enter at least one object file.

## Linker Input and Output

---

### Specifying Executable Output Type

You can use the linker to produce executable modules (with the extension `.exe`) and dynamic link libraries (with the extension `.dll`). The linker produces `.exe` files by default.

Use options to specify what kind of output you want:

- To produce an `.exe`, specify the `/EXEC` option.
- To produce a `.DLL`, specify the `/DLL` option. Instead, you can specify the `/Ge-` compiler option if you are using the compiler to invoke the linker.

### Producing an `.exe` File

The linker produces `.exe` files by default. Use the `/EXEC` option to explicitly identify the output file as an `.exe` file.

An `.exe` file is one that can be executed directly: you can run the program by typing the name of the file. In contrast, `DLL` programs execute when they are called by other processes, and cannot be run independently.

To reduce the size of the `.exe` file and improve its performance, use the following options:

- `/ALIGNFILE:n` to set the file alignment for sections in the output file. Set `n` to smaller factors to reduce the size of the executable, and to larger factors to reduce load time for the executable. By default, the alignment is set to 512.
- `/BASE:n` to specify the load address for the executable. For example, if several `DLLs` are loaded at base addresses that ensure that the `DLLs` do not overlap, the linker does not have to reapply the relocation records.
- `/OPTFUNC` (once your code is tested and stable) to remove unreachable functions.


**Note:** Your code must have been compiled with `G1+`.

If you do not specify an extension for the output file name, the linker automatically adds the extension `.exe` to the name you provide. If you do not specify an output filename at all, the linker generates an `.exe` file with the same filename as the first `.OBJ` file it linked.

## Linker Input and Output

### Producing a Dynamic Link Library


A dynamic link library (.dll) file contains executable code for common functions, just as a static library (.lib) file does. However, when you link with a DLL (using an import library), the code in the DLL is **not** copied into the executable file. Instead, only the import definitions for DLL functions are copied, resulting in a smaller executable. At run time, the dynamic link library is loaded into memory, along with the .exe file.

To produce a DLL as output, compile at least one object file (that does not include the main function) with the /Ge- compiler option, and link them with the /DLL linker option. You must include an export definition (.exp) file that specifies which functions are to be included in the DLL.  see “Controlling ILIB Output” on page 523.

To reduce the size of the DLL and improve its performance, use the following options:

- /ALIGNFILE:value to set the alignment factor in the output file. Set *value* to smaller factors to reduce the size of the DLL, and to larger factors to reduce load time for the DLL. By default, the alignment is set to 512.
- /OPTFUNC (once your code is tested and stable) to remove unreachable functions.

For DLLs, setting a /BASE value can save load time when the given load address is available. If the load address is not available, the /BASE value is ignored, and there is no load time benefit.

Once you have produced the DLL, you can produce an executable that links to the DLL. Use the ILIB utility to create an import library, and then use the .lib file as input to the linker.  See Chapter 45, “Using IBM Library Manager” on page 519.


---

### Generating a Map File

Specify /MAP to generate a map file, which lists the object modules in your output file; section names, addresses, and sizes; and symbol information. If you do not specify a name for the map file, the map file takes the name of the executable output file, with the extension .map.

To prevent the map file from being generated, specify the /NOMAP option.

Specify /LINENUMBERS to include source file line numbers and associated addresses in the map file.

 See “Linker Options” on page 202 for more information on these and other options.

## Linker Input and Output

---

### Linker Return Codes

The linker has the following return codes:


Code	Meaning
------	---------

- |    |                                                                                                           |
|----|-----------------------------------------------------------------------------------------------------------|
| 0  | The link was completed successfully. The linker detected no errors, and issued no warnings.               |
| 4  | <b>Warnings</b> issued. There may be problems with the output file.                                       |
| 8  | <b>Errors</b> detected. The linking might have completed, but the output file cannot be run successfully. |
| 12 | Both warnings issued and errors detected (see return codes 4 and 8)                                       |
| 16 | <b>Severe errors</b> detected. Linking ended abnormally, and the output file cannot be run successfully.  |
| 20 | Both warnings issued and severe errors detected (see return codes 4 and 16)                               |
| 24 | Both errors and severe errors issued (see return codes 8 and 16)                                          |
| 28 | The linker issued warnings, detected errors, and detected severe errors (see return codes 4, 8, and 16)   |

If you invoke the linker through the compiler, then a return code of zero is issued for warnings. If you invoke the linker through a makefile, you can force NMAKE to ignore warnings by putting -7 before the ILINK command.



## Chapter 12. Linking with Library Files

The linker searches library files to resolve references in your object files to functions and data outside the object files. The library files contain object modules, which contain the code for functions your object files can reference. Some libraries are searched by default: the compiler embeds the names of default libraries into object files.  See “Choosing Your Runtime Libraries” on page 91 for more information on choosing your default libraries.

There are two kinds of library files: static libraries, and dynamic link libraries (DLLs) (which you access through import libraries). The default libraries are static library files, unless you specify the compiler option `/Gd` so you can link to the DLL versions of the default libraries. In static linking, you link to `.lib` files that contain the object modules you need.

**Static linking** means that code for all the functions called in your program is copied from a `.lib` file into your output `.exe` or `.dll` file. The `.exe` or `.dll` files will be larger because there is a copy of the runtime functions in each file. These programs will take up more storage, and if you run them at the same time, there will also be a copy of the library functions in memory for each program. Statically linked programs, however, are easier to distribute because the library functions are part of your executable file.

**Dynamic linking** means that code for the VisualAge for C++ runtime functions called in your program is **not** copied into your output `.exe` or `.dll` file. Instead, the function code stays in a separate VisualAge for C++ DLL file, and your calls to the function are resolved at load time. The amount of disk space required by your `.exe` or `.dll` file is reduced, and there is only one copy of the library functions in memory for all programs that use them. Dynamically linked programs can be harder to distribute, since the separate DLL file must be distributed along with your executable file. They can also be easier to maintain, because you do not necessarily need to relink whenever there is a change in one of the DLLs you use.

Use the `/Gd` compiler option to control whether your executable file links to the runtime library statically or dynamically. The default is `/Gd-`, which statically links with the `.lib` version of the runtime library. Specify `/Gd+` to dynamically link with the DLL version of the runtime library.

The compiler option you choose causes the corresponding library to be linked in by default. If you override the default libraries with the `/NOD` linker option, you must explicitly give the name of all libraries you are using on the linker command line.

## Linking with .lib Files • Linking to DLLs

If you are linking with libraries other than the default libraries, you can specify the libraries when you link. The libraries you specify at the linking step are searched before the default libraries.

---

### Linking with .lib Files

The linker uses library (.lib) files to resolve external references from code in the object (.OBJ) files. When the linker finds a function or data reference in a .obj file that requires a module from a library, the linker links the module from the library to the output file.



The compiler embeds the names of needed libraries (called default libraries) in object files. You do not need to specify these libraries: the linker searches them automatically.

Specify library files only when:

- You want to use libraries other than the default libraries. Libraries you specify are searched ahead of the default libraries.
- You want to specify the default library with its full path, because the default library is not in the current directory, and not in a directory specified in the LIB environment variable (see “Search Rules” on page 187 for more information). You can also specify a path to a directory, without specifying a file name, to have the linker search that directory before the directories in the LIB environment variable (see “Specifying Directories” on page 188 for more information).

To ignore the default libraries, use the /NODEFAULTLIBRARYSEARCH option. Use the option with care, because most compilers expect their object files to be linked with default libraries.

---

### Linking to Dynamic Link Libraries

A dynamic link library (DLL) file contains executable code for common functions, just as an ordinary library (.lib) file does. However, when you link with a DLL, the code in the DLL is **not** copied into the executable (.exe) file. Instead, only the import definitions for DLL functions are copied. At run time, the dynamic link library that contains the functions is loaded into memory, along with the .exe file that references them.

To link with a DLL, include the import library (.lib) file for the DLL on the linker command line. The import library allows the linker to use data or functions exported from the DLL to resolve references in your object code. When you run your application, the references resolve to the DLL.



## Linking to DLLs

### Advantages of Using DLLs

#### Size

Applications require less disk space. With dynamic linking, applications that use the same function can share a single copy of the function, rather than each application having its own copy of the function.

#### Independence

Dynamic link libraries and applications stay independent. You can update dynamic link libraries without relinking the applications that use them (as long as the function names and parameters are preserved). If you use third-party DLLs, this is particularly convenient, because you can update the third-party DLL without recompiling or relinking your programs. At run time, your applications automatically call the updated function code from the new DLL.

#### Sharing

When appropriate, code and data sections loaded from a dynamic link library can be shared by the different applications that access the DLL. Without dynamic linking, such sharing is not possible because each file has its own copy of all the code and data it uses. By sharing sections with dynamic linking, applications can use memory more efficiently. However, use caution when sharing data sections in order to avoid multiple users trying to update the same section concurrently.


#### Linking Speed

Your applications link more quickly, because the linker does not have to copy code from the library into your program.

## Linking to DLLs



## Chapter 13. Setting Linker Options


Linker options are not case sensitive, so you can specify them in lowercase, uppercase, or mixed case. You can also substitute a dash (-) for the slash (/) preceding the option. For example, -DEBUG is equivalent to /DEBUG. You can specify options in either a short or long form. For example, /DE, /DEB, and /DEBU are all equivalent to /DEBUG.  See “Summary of Linker Options” on page 201 for the shortest acceptable form for each option. Lowercase and uppercase, short and long forms, dashes, and slashes can all be used on one command line, as in:

```
ilink /de -DBGPACK -Map /DLL prog.obj
```

Separate options with a space or tab character. You can specify linker options in the following ways:


- On the command line
- In the ILINK environment variable
- In WorkFrame

Options specified on the command line override the options in the ILINK environment variable. Options specified in the ILINK environment variable override settings specified by directives, including many statements in a module definition file.

Some linker options take numeric arguments. You can enter numbers in decimal, octal, or hexadecimal format using standard C-language syntax.  See “Specifying Numeric Arguments” on page 199 for more information.

If conflicting options are specified (e.g., /DEBUG and /NODEBUG), the last one specified is used.

### Setting Options on the Command Line

Linker options specified on the command line override any previously specified in the ILINK environment variable (as  described in “Setting Options in the ILINK Environment Variable” on page 198).

You can specify options anywhere on the command line. Separate options with a space or tab character.

For example, to link an object file with the /MAP option, enter:


```
ilink /M myprog.obj
```

## Setting Linker Options

### Setting Options in the ILINK Environment Variable

Store frequently used options in the ILINK environment variable. This method is useful if you find yourself repeating the same command-line options every time you link. You cannot specify filenames in the environment variable, only linker options.

The ILINK environment variable can be set from the command line or in a command (.cmd) file. In Windows 95, it can also be set in the autoexec.bat file. In Windows NT, it can also be set in the **System** window (to get there, double-click on **Main** and then on **Control Panel**). In the **System** window, click on **Set** to add a new item to the list of User Environment Variables. If it is set on the command line or by running a command file, the options will only be in effect for the current session (until you reboot your computer). If it is set in the AUTOEXEC.BAT file or the Windows NT **System** window, the options are set when you boot your computer, and are in effect every time you use the linker unless you override them using a .cmd file or by specifying options on the command line.

 See “Windows Environment Variables for Compiling” on page 66 for more information about using ILINK and other environment variables.

#### Example

In the following example, options on the command line override options in the environment variable.



```
SET ILINK=/DLL /DE
ILINK test
ILINK /NODEF /NODEB prog
```

The first command sets the environment variable to the options /DLL and /DEBUG.

The second command links the file test.obj, using the options specified in the environment variable, to produce test.dll.


The last command links the file prog.obj to produce prog.dll, using the option /NODEFAULTLIBRARYSEARCH, in addition to the /DLL option. The /NODEBUG option on the command line overrides the /DEBUG option in the environment variable, and the linker links without the /DEBUG option.

### Setting Options in the WorkFrame Environment

If you have installed the WorkFrame product, you can set linker options using the options dialogs. You can use the dialogs when you create or modify a project.

Options you select while creating or changing a project are saved with that project.

## Setting Linker Options

For more information on setting options and linking with WorkFrame,  see “Linking within WorkFrame” on page 179.

In this guide, linker options are generally referred to in their command-line form. The following summary maps the command-line forms to the WorkFrame Link options notebook forms:

OPTION	NOTEBOOK TAB	PAGE	CONTROL DESCRIPTION
/ALIGNADDR	Generation	3	Segment alignment
/ALIGNFILE	Generation	3	File alignment
/BASE	Generation	2	Module load address
/BROWSE	Generation	1	Include browse information
/CODE	Definition	1	Code attributes
/DATA	Definition	1	Data attributes
/DBGPACK	Generation	1	Pack debug information
/DEBUG	Generation	1	Include debug information
/DEFAULTLIB	File Names	1	Default library searching
/DLL	Definition	3	Run-file
/EXEC	Definition	3	Run-file
/EXTDICTIONARY	Processing	1	Search the extended libraries
/HEAP	Generation	1	Heap size
/INCLUDE	Processing	1	Force a reference to symbol
/LINENUMBERS	Processing	1	Generate a map file with line numbers
/LOGO	Processing	1	Don't display the logo
/MAP	File Names	1	Map (.map) file name
/MAP	Processing	1	Generate a map file
/OPTFUNC	Generation	3	Removed unreferenced functions
/OUT	File Names	1	Generated target file name
/PMTYPE	Generation	1	Application type
/SECTION	Definition	2	Memory protection
/SEGMENTS	Generation	3	Max # of sections
/STACK	Generation	1	Stack size
/STUB	Definition	1	User-chosen DOS stub
/SUBSYSTEM	Generation	2	Subsystem
/VERBOSE	Processing	1	Display link-time information
/VERSION	Definition	1	User-chosen version number

---

### Specifying Numeric Arguments

Some linker options and module statements take numeric arguments. The linker accepts numeric arguments in C-language syntax. You can specify numbers in any of the following forms:

**Decimal** Any number **not** prefixed with 0 or 0x is a decimal number. For example, 1234 is a decimal number.

**Octal** Any number prefixed with 0 (but not 0x) is an octal number. For example, 01234 is an octal number.

## Setting Linker Options

**Hexadecimal** Any number prefixed with 0x is a hexadecimal number. For example, 0x1234 is a hexadecimal number.

## Linker Options Summary

### Summary of Linker Options

Figure 26 (Page 1 of 2). Linker Options Summary

Syntax	Description	Default	Page
/?	Display help	None	202
/ALIGN[ADDR]: <i>factor</i>	Set address alignment	/A:0x00010000	203
/A[LIGNFILE]: <i>factor</i>	Set file alignment	/A:512	203
/BAS[E]: < <i>address</i> @ <i>filename</i> , <i>key</i> >	Set preferred loading address	/BAS:0x00400000	203
/BR[OWSE] /NOBR[OWSE]	Add browse information	/NOBR	204
/CODE: <i>attributes</i>	Set section attributes for executable	/CODE:RX	204
/DATA: <i>attributes</i>	Set section attributes for data	/DATA:RW	205
/DB[GPACK] /NODB[GPACK]	Pack debugging information	/NODB	205
/D[EBUG] /NODEB[UG]	Include debugging information	/NODEB	206
/DEF[AULTLIBRARYSEARCH][: <i>lib</i> ] /NOD[EFAULTLIBRARYSEARCH][: <i>lib</i> ]	Search default libraries	/DEF	206
/DLL	Generate DLL	/EXEC	207
/EN[TRY]: <i>name</i>	Specify an entry point in an executable file	None	207
/EXEC[UTABLE]	Generate .exe file	/EXEC	207
/EXT[DICTIONARY] /NOE[XTDICTIONARY]	Use extended dictionary to search libraries	/EXT	208
/FI[xed] /NOFI[xed]	Do not relocate the file in memory	/NOFI	208
/FO[RCE] /NOFO[RCE]	Create executable output file even if errors are detected	/NOFO	209
/HEA[P]: <i>reserve</i> [, <i>commit</i> ]	Set size of program heap	/HEAP: 0x100000,0x1000	209
/H[ELP]	Display help	None	209
/INC[lude]	Forces a reference to a symbol	None	209
/I[NFORMATION] /NOIN[FORMATION]	Display status of linking process	/NOIN	210
/L[INENUMBERS] /NOLI[NENUMBERS]	Include line numbers in map file	/NOLI	210
/LO[GO] /NOL[OGO]	Display logo, echo response file	/LO	210
/M[AP][: <i>name</i> ] /NOM[AP]	Generate map file	/NOM	211

## **/? Option •/A Option**

Figure 26 (Page 2 of 2). Linker Options Summary

Syntax	Description	Default	Page
/OPTF[UNC] /NOOPTF[UNC]	Remove unreferenced functions	/NOOPTF	211
/O[UT][:name]	Name output file	Name of first .obj file	212
/PM[TYPE]:type	Specify application type	PMTYPE:VIO	212
/SEC[TION]:name, attributes	Set attributes for section	Set by /CODE and /DATA	213
/SE[GMENTS]:number	Set maximum number of segments	/SE:256	213
/ST[ACK]:reserve [,commit]	Set stack size of application	/STACK: 0x100000,0x1000	214
/STU[B]:filename	Specify the name of the DOS stub file	None	214
/SU[BSYSTEM]:subsystem [,major[.minor]]	Specify the required subsystem and version	/SUBSYSTEM: WINDOWS,4.0	215
/VERB[OSE] /NOV[ERBOSE]	Display status of linking process	/NOV	215
/VER[SION]:major [.minor]	Write a version number in the run file	/VERSION:0.0	215

## **Linker Options**

### **/?**

**Syntax:**  
/?

**Default:**  
None

Use /? to display a list of valid linker options. This option is equivalent to /HELP.

### **/ALIGNADDR**

**Syntax:**  
/ALIGN[ADDR]:factor

**Default:**  
/ALIGNADDR:0x00010000

Use /ALIGNADDR to set the address alignment for segments.

The alignment factor determines where segments in the .exe or .dll file start. From the beginning of the file, the start of each segment is aligned at a multiple (in bytes) of the alignment factor. The alignment factor must be a power of 2, from 512 to 256M.



## /A Option •/BAS Options

The default alignment is 64K bytes.

### /ALIGNFILE

**Syntax:**

/A[LIGNFILE]:*factor*

**Default:**

/ALIGNFILE:512

Use /ALIGNFILE to set the file alignment for segments.

The alignment factor determines where segments in the .exe or .dll file start. From the beginning of the file, the start of each segment is aligned at a multiple (in bytes) of the alignment factor. The alignment factor must be a power of 2, from 512 to 64K.

The default alignment is 512 bytes.

### /BASE

**Syntax:**

/BAS[E]: <*address*@*filename*, *key*>

**Default:**

/BASE:0x00400000

Use /BASE to specify the preferred load address for the first load segment of a .DLL file.

Specifying @*filename*, *key*, in place of *address*, bases a set of programs (usually a set of DLLs) so they do not overlap in memory. *filename* is the name of a text file that defines the memory map for a set of files. *key* is a reference to a line in *filename* beginning with the specified key. Each line in the memory-map file has the syntax:

*key address maxsize*

Separate the elements with one or more spaces or tabs. The *key* is a unique name in the file. The *address* is the location of the memory image in the virtual address space. The *maxsize* is an amount of memory within which the image must fit. The linker will issue a warning when the memory image of the program exceeds the specified size. A comment in the memory-map file begins with a semicolon (;) and runs to the end of the line.

## **/BR, /NOBR Options •/CODE Option**

### **/BROWSE, NOBROWSE**

**Syntax:**  
/BR[OWSE]  
/NOBR[OWSE]

**Default:**  
/NOBROWSE

Use /BROWSE to add browse information to the load module the linker generates. /BROWSE is automatically turned on when you specify /DEBUG. This option is only effective if the object files are compiled with the /Fb or /Fb\* compiler option.

The browse information is used by the VisualAge for C++ Browser when you browse your program.

See “Creating Files to Use with the Browser” on page 335 for more information.

If you are compiling and linking in one step, specifying /Fb automatically passes /BROWSE to the linker.

### **/CODE**

**Syntax:**  
/CODE:*attributes*

**Default:**  
/CODE:RX

Use /CODE to specify the default attributes for all code sections. Letters can be specified in any order.

<b>Letter</b>	<b>Attribute</b>
<b>E or X</b>	EXECUTE
<b>R</b>	READ
<b>S</b>	SHARED
<b>W</b>	WRITE

The default is /CODE:RX.

## /DATA Option •/DB, /NODB Options

### /DATA

**Syntax:**

/DATA:*attributes*

**Default:**

/DATA:RW

Use /DATA to specify the default attributes for all data sections. Letters can be specified in any order.

Letter	Attribute
E or X	EXECUTE
R	READ
S	SHARED
W	WRITE

The default is /DATA:RW.

### /DBGPACK, /NDBGPACK

**Syntax:**

/DB[GPACK]  
/NODB[GPACK]

**Default:**

/NDBGPACK

Use /DBGPACK to eliminate redundant debug type information. The linker takes the debug type information from all object files and needed library components, and reduces the information to one entry per type. This results in a smaller executable output file, and can improve debugger performance.

**Performance Consideration:** Generally, linking with /DBGPACK slows the linking process, because it takes time to pack the information. However, if there is enough redundant debug type information, /DBGPACK can actually speed up your linking, because there is less information to write to file.

When you specify /DBGPACK, /DEBUG and /BROWSE are turned on by default.

## **/D, /NODEB Options •/DEF, /NOD Options**

### **/DEBUG, /NODEBUG**

**Syntax:**  
/D[EBUG]  
/NODEB[UG]

**Default:**  
/NODEBUG


Use /DEBUG to include debug information in the output file, so you can debug the file with the debugger, or analyze its performance with the Performance Analyzer. The linker will embed symbolic data and line number information in the output file.

For debugging, compile the object files with /Ti.

For the Performance Analyzer, compile the object files with /Ti and /Gh.

When you specify /DEBUG, /BROWSE is turned on by default.

**Note:** Linking with /DEBUG increases the size of the executable output file.

 See Part 5, “Debugging Your Program” on page 261 for more information.

### **/DEFAULTLIBRARYSEARCH, /NODEFAULTLIBRARYSEARCH**

**Syntax:**  
/DEF[AULTLIBRARYSEARCH] [:*library*]  
/NOD[EFAULTLIBRARYSEARCH] [:*library*]

**Default:**  
/DEFAULTLIBRARYSEARCH



Use /DEFAULTLIBRARYSEARCH to have the linker search the default libraries of object files when resolving references.

If you specify a *library* with the option, the linker adds the library name to the list of default libraries. The default libraries for an object file are defined at compile time, and embedded in the object file. The linker searches the default libraries by default.

Use /NODEFAULTLIBRARYSEARCH to tell the linker to ignore default libraries when it resolves external references. If you specify a *library* with the option, the linker ignores that default library, but searches the rest of the default libraries (and any others that are defined in the object files).

If you specify /NODEFAULTLIBRARYSEARCH without specifying *library*, then you must explicitly specify all the libraries you want to use, including VisualAge for C++ runtime libraries.

## **/DLL Option •/EXEC Option**


 See “Choosing Your Runtime Libraries” on page 91 for more information on defining the default libraries when you compile, and  see “Linking with .lib Files” on page 194 for more information on explicitly specifying libraries when you link.

If you are compiling and linking in one step, specifying /Gn automatically passes /NODEFAULTLIBRARYSEARCH to the linker.

### **/DLL**

**Syntax:**  
/DLL

**Default:**  
/EXEC

Use /DLL to identify the output file as a dynamic link library (.dll file). For more information on generating a DLL, see  “Producing a Dynamic Link Library” on page 191. The object files should be compiled with /Ge-

If you specify /DLL with /EXEC, only the last specified of the options takes effect.

If you do not specify /DLL, or any of the other options above, then by default the linker produces an .exe file (/EXEC).

### **/ENTRY**

**Syntax:**  
/ENTRY :*name*


**Default:**  
None

Use /ENTRY to specify an entry point (name of a routine or function) in an executable.

### **/EXECUTABLE**

**Syntax:**  
/EXEC[UTABLE]

**Default:**  
/EXEC

Use /EXEC to identify the output file as an executable program (.EXE file). The linker generates .exe files by default. For more information on generating an .exe file,  see “Producing an .exe File” on page 190. The object files should be compiled with /Ge+.

If you specify /EXEC with /DLL, only the last specified of the options takes effect.

If you do not specify /EXEC, or any of the other options above, then the linker produces an .exe file by default.

## **/EXT, /NOE Options • /FI, /NOFI Options**

### **/EXTDICTIONARY, /NOEXTDICTIONARY**

**Syntax:**

/EXT[DICTIONARY]  
/NOE[XTDICTIONARY]

**Default:**

/EXTDICTIONARY

Use /EXTDICTIONARY to have the linker search the extended dictionaries of libraries when it resolves external references. The extended dictionary is a list of module relationships within a library. When the linker pulls in a module from the library, it checks the extended dictionary to see if that module requires other modules in the library, and then pulls in the additional modules automatically.

The linker searches the extended dictionary by default, to speed up the linking process.

Use /NOEXTDICTIONARY if you are defining a symbol in your object code that is also defined in one of the libraries you are linking to. Otherwise the linker issues an error because you have defined the same symbol in two different places. When you link with /NOEXTDICTIONARY, the linker searches the dictionary directly, instead of searching the extended dictionary. This results in slower linking, because references must be resolved individually.

### **/FIXED, /NOFIXED**

**Syntax:**

/FI[XED]  
/NOFI[XED]

**Default:**

/NOFIXED

Use /FIXED to tell the loader not to relocate a file in memory when the specified base address is not available.

For more information on base addresses, see the /BASE linker option.

By default, the linker uses /NOFIXED.

## /FO Option •/INC Option

### /FORCE

**Syntax:**  
/FO[RCE]  
/NOFO[RCE]

**Default:**  
/NOFO

Use /FORCE to produce an executable output file even if there are errors during the linking process.

By default, the linker does not produce an executable output file if it encounters an error.

### /HEAP

**Syntax:**  
/HEA[P]:*reserve*[,*commit*]

**Default:**  
/HEAP:0x100000,0x1000

Use /HEAP to set the size of the program heap in bytes. The *reserve* argument sets the total virtual address space reserved. The *commit* sets the amount of physical memory to allocate initially. When *commit* is less than *reserve*, memory demands are reduced, but execution time can be slower.

### /HELP

**Syntax:**  
/H[ELP]

**Default:**  
None

Use /HELP to display a list of valid linker options. This option is equivalent to /?.

### /INCLUDE

**Syntax:**  
/INC[LUDE]:*symbol*

**Default:**  
None

Use /INCLUDE to force a reference to a symbol. The linker searches for an object module that defines the symbol.

## **/I, /NOIN Options •/LO, /NOL Options**

### **/INFORMATION, /NOINFORMATION**

**Syntax:**

/I[NFORMATION]  
/NOIN[FORMATION]

**Default:**

/NOINFORMATION

See the description of the /VERBOSE linker option.

### **/LINENUMBERS, /NOLINENUMBERS**

**Syntax:**


/L[LINENUMBERS]  
/NOLI[NENUMBERS]

**Default:**

/NOLINENUMBERS

Use /LINENUMBERS to include source file line numbers and associated addresses in the map file. For this option to take effect, there must already be line number information in the object files you are linking. When you compile, use the /Tn option to include line numbers in the object file (or the /Ti option, to include all debugging information). If you give the linker an object file without line number information, the /LINENUMBERS option has no effect.

The /LINENUMBERS option forces the linker to create a map file, even if you specified /NOMAP.

For more information on map files,  see “Generating a Map File” on page 191.

By default, the map file is given the same name as the output file, plus the extension .map. You can override the default name by specifying a map filename.

### **/LOGO, /NOLOGO**

**Syntax:**

/LO[GO]  
/NOL[OGO]

**Default:**

/LOGO

Use /NOLOGO to suppress the product information that appears when the linker starts.

Specify /NOLOGO before the response file on the command line, or in the ILINK environment variable. If the option appears in or after the response file, it is ignored.

If you are compiling and linking in one step, you can use the /Q compiler option to suppress the product information, and stop the contents of the response file from being echoed to the screen.



## /M, /NOM Options • /OPTF, NOOPTF Options

By default, the linker displays product information at the start of the linking process, and displays the contents of the response file as it reads the file.

### /MAP, /NOMAP

**Syntax:**


/M[AP] [:*name*]  
/NOM[AP]

**Default:**

/NOM

Use /MAP to generate a map file called *name*. The file lists the composition of each segment, and the public (global) symbols defined in the object files. The symbols are listed twice: in order of name and in order of address.

If you do not specify a directory, the map file is generated into the current working directory. If you do not specify *name*, the map file has the same name as the executable output file, with the extension .map.

For more information on map files,  see “Generating a Map File” on page 191.

If you are compiling and linking in one step, you can use the /Fm compiler option to generate a linker map file.

By default, the linker does not produce a map file.

### /OPTFUNC, /NOOPTFUNC

**Syntax:**

/OPTF[UNC]  
/NOOPTF[UNC]


**Default:**

/NOOPTF

Use /OPTFUNC to remove unreachable functions. The linker removes functions that are:



- Not referenced anywhere in the object code
- Rendered unreferenced by the removal of other functions
- Not exported for use in other files

 See “EXPORTS” on page 537 for more information on exporting functions.

When the function is removed, any additional functions that were required only by that function are also removed. Removing the functions and code reduces the size of your .exe or .dll output file.

By default, the linker does not remove unreachable functions.

## /OUT Option • /PM Option

**Note:** The object code you are linking must be compiled with the /G1 option for /OPTFUNC to take effect.

If you are compiling and linking in one step, you do not need to specify /OPTFUNC: the compiler passes /OPTFUNC to the linker automatically when you specify /G1.

**Performance Consideration:** Optimized linking generally takes longer than regular linking, because of the extra processing that the linker performs. However, if the optimization is effective enough, it can actually speed up the linking process, because there is less information to write to file. Generally, you may want to link without the /OPTFUNC option until your code is tested and stable.

## /OUT

**Syntax:**  
/O[UT][:*name*]

**Default:**  
Name of first .obj file with appropriate extension

Use /OUT to specify a name for the executable output file.

If you do not provide an extension with *name*, then the linker provides an extension based on the type of file you are producing:

File produced	Default extension
Executable program	.exe
Dynamic link library	.dll

If you do not use the /OUT option, then the linker uses the filename of the first object file you specified, with the appropriate extension.

## /PMTYPE

**Syntax:**  
/PM[TYPE]:*type*

**Default:**  
/PMTYPE:VIO

Use /PMTYPE to specify the type of .exe file that the linker generates. Do not use this option when generating dynamic link libraries (DLLs).

One of the following types must be specified:

- PM** The executable must be run in a window.
- VIO** The executable can be run either in a window or in a full screen.
- NOVIO** The executable must not be run in a window; it must use a full screen.

## /SECTION

**Syntax:**  
`/SEC[TION]:name,attributes`

**Default:**  
Depends on section type

Use `/SECTION` to specify memory-protection attributes for the *name* section. *name* is case sensitive. You can specify the following attributes:

Letter	Sets Attribute
<b>E</b> or <b>X</b>	EXECUTE
<b>R</b>	READ
<b>S</b>	SHARED
<b>W</b>	WRITE

For example,  
`/SEC:dseg1,RS`

sets the READ and SHARED attributes, but not the EXECUTE, or WRITE attributes, for the section `dseg1` in an `.exe` file.

### Defaults




Sections are assigned attributes by default, as follows:

Segment	Default Attributes
Code sections	EXECUTE, READ (ER)
Data sections (in <code>.exe</code> file)	READ, WRITE (RW), not shared
Data sections (in <code>.dll</code> file)	READ, WRITE, not shared
CONST32_RO section	READ, SHARED (RS)

## /SEGMENTS

**Syntax:**  
`/SE[GMENTS]:number`

**Default:**  
`/SE:256`

Use `/SEGMENTS` to set the number of logical segments a program can have. You can set *number* to any value in the range 1 to 16375.  See “Specifying Numeric Arguments” on page 199.

For each logical segment, the linker must allocate space to keep track of segment information. By using a relatively low segment limit as a default (256), the linker is able to link faster and allocate less storage space.

## /ST Option •/STU Option

When you set the segment limit higher than 256, the linker allocates more space for segment information. This results in slower linking, but allows you to link programs with a large number of segments.

For programs with fewer than 256 segments, you can improve link time and reduce linker storage requirements by setting *number* to the actual number of segments in the program.

## /STACK

**Syntax:**


/ST[ACK]:*reserve* [,*commit*]

**Default:**

/STACK:0x100000,0x1000

Use /STACK to set the stack size (in bytes) of your program. The size must be an even number from 0 to 0xFfffffe. If you specify an odd number, it is rounded up to the next even number.

*reserve* indicates the total virtual address space reserved. *commit* sets the amount of physical memory to allocate initially. When *commit* is less than *reserve*, memory demands are reduced, although execution time may be slower.

 See “Controlling Stack Allocation and Stack Probes” on page 102 for more information.

## /STUB

**Syntax:**

/STU[B]:*filename*

**Default:**

None

Use /STUB to specify the name of the DOS executable at the beginning of the output file created.

By default, the linker defines its own stub.

## /SU Option •/VER Option

### /SUBSYSTEM

**Syntax:**

/SU[BSYSTEM]: *subsystem*[*major*[*minor* ]]

**Default:**

/SUBSYSTEM:WINDOWS,4.0

Use /SUBSYSTEM to specify the subsystem and version required to run the program. The *major* and *minor* arguments are optional and specify the minimum required version of the subsystem. The *major* and *minor* arguments are integers in the range 0 to 65535.

Subsystem	Major.Minor	Description
WINDOWS	3.10	A graphical application that uses the Graphical Device Interface (GDI) API.
CONSOLE	3.10	A character-mode application that uses the Console API.

### /VERBOSE

**Syntax:**

/VERB[OSE]  
/NOV[ERBOSE]

**Default:**

/NOVERBOSE

Use /VERBOSE to have the linker display information about the linking process as it occurs, including the phase of linking and the names and paths of the object files being linked.

If you are having trouble linking because the linker is finding the wrong files or finding them in the wrong order, use /VERBOSE to determine the locations of the object files being linked and the order in which they are linked.

The output from this option is sent to **stdout**. You can redirect the output to a file using Windows redirection symbols.

/VERBOSE is the same as /INFORMATION.

### /VERSION

**Syntax:**

/VER[SION]: *major*[*minor*]

**Default:**

/VERSION:0.0

Use /VERSION to write a version number in the header of the run file. The *major* and *minor* arguments are integers in the range 0 to 65535.

## **/VER Option**

---

## Part 4. Editing Files

The following tasks will familiarize you with the VisualAge for C++ editor. You can use this editor to edit C and C++ files. For more help with using editor commands, see the online *Editor Command Reference*.

---

<b>Chapter 14. Introduction to the Editor</b> . . . . .	219
Creating a New File . . . . .	219
Entering and Editing Text . . . . .	220
Undoing Changes . . . . .	223
Saving a File . . . . .	224
Closing a Document View . . . . .	225
Opening an Existing File . . . . .	225
Finding Text . . . . .	226
Finding and Replacing Text . . . . .	227
Finding a Specific Line in the File . . . . .	228
Creating and Finding Marks . . . . .	229
Embedding Another File in the Current Document . . . . .	232
Issuing Editor Commands . . . . .	233
Blocking and Manipulating Text . . . . .	235
Working with Multiple Document Views . . . . .	239
Using a Parser . . . . .	246
 <b>Chapter 15. Customizing the Editor</b> . . . . .	 249
Using the Editor Tool Bar . . . . .	249
Adding Your Own Items to the Menu Bar . . . . .	251
Customizing the Keyboard . . . . .	252
Customizing the Autosave Facility for the Editor . . . . .	253
Changing Editor Tab Settings . . . . .	255
Changing the Base Editor Font . . . . .	256
Changing Token Display Attributes . . . . .	257
Modifying Editor Behavior Permanently . . . . .	258

---





## Chapter 14. Introduction to the Editor

This chapter introduces you to some of the basic features of the Editor. Examples in this section often build on each other, so they should be performed in the order they are given.

As you perform the suggested exercises in this section, pay close attention to the various command menus you see. Many actions on these menus can also be accessed directly from the keyboard with special keystroke combinations. Such keystroke combinations appear to the right of the menu choices to which they apply.

---

### Creating a New File

You can use the Editor to create a new file in many different ways:

- If it is not already running, start the Editor by doing one of the following:
  - Double-click on the Editor icon in an application folder. An empty Editor window similar to that shown in Figure 28 on page 220 appears.
  - Type `iedit` at a command line prompt.
- If the Editor is already running, do the following:
  1. Select **File** from the Editor menu bar.
  2. Select **New...** from the resulting pull-down menu. The window shown in Figure 27 appears.

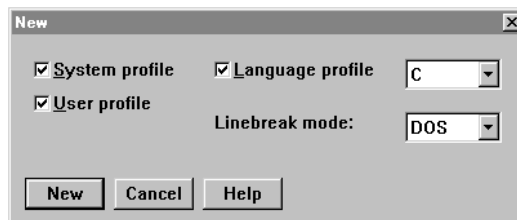


Figure 27. New Window

3. Use the check boxes and selection lists to choose the options you want. For example, in Figure 27, the C-language profile has been selected, which instructs the Editor to load a C-language parser. The Editor also comes with many other profiles that help you use color and text emphasis to highlight elements of your document. For more information, see “Using a Parser” on page 246.

## Editor Introduction

4. Click on the **New** push button in this window. A new, empty Editor window similar to that shown in Figure 28 on page 220 appears.

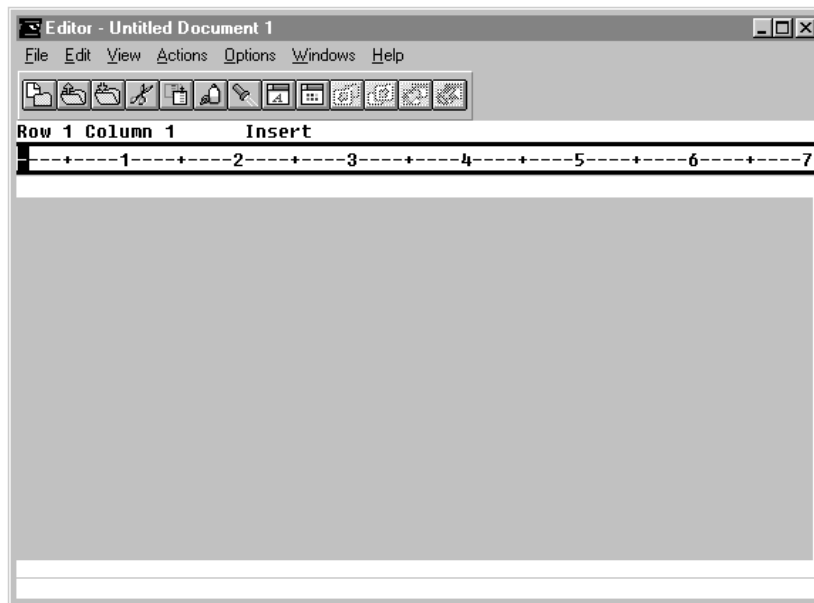


Figure 28. Editor Window

You can start entering text for your new file in this window.

---

## Entering and Editing Text

This section shows you how to enter text in the editing window. This section also introduces you to some of the special keys you can use to edit text. For more information on the keys available for a window, select **Help** from the menu bar, then select **Keys help** from the resulting pull-down menu.

**Entering Text** To enter text, just begin typing. Type the following text in the editor:

```
/* This is only the beginning! */  
/* And for now, this is the end. */
```

If you make a typing error, use the Backspace key to go back and correct the error. When you are done, your editing window will look similar to Figure 29 on page 221.

## Editor Introduction

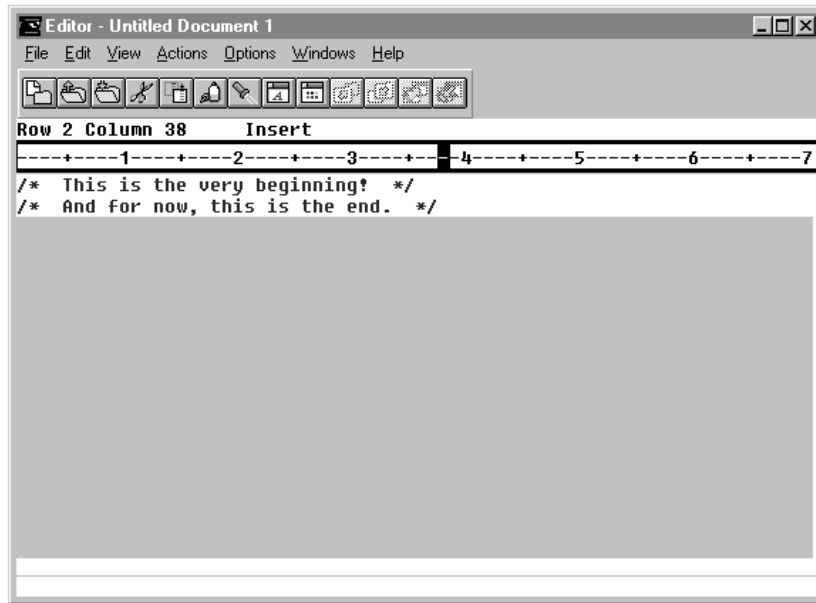


Figure 29. Editor Window with New Text

### Moving the Cursor

Use any of the following methods to move the cursor in the editing window:

- Position the mouse pointer where you want the cursor to be, and click mouse button one. If you inadvertently select text by dragging the mouse, deselect the text by pressing Alt+U or by clicking mouse button two.
- Press the Up, Down, Left, or Right Arrow keys.
- Press the Home key to move the cursor to the beginning of a line, or the End key to move it to the end of a line.
- Press Alt+Left arrow to move the cursor one word left, or Alt+Right arrow to move it one word right.
- Press the Page Up or Page Down keys to move the cursor up or down one window at a time.
- Press Ctrl+Home to move the cursor to the beginning of a file, or Ctrl+End to move it to the end of a file.
- Press Ctrl+J to move the cursor to the place where you last entered text.

### Inserting a Blank Line

To create a new line in an editing window, do one of the following:

- Move the cursor to any point on a line. Press Ctrl+Enter.
- Move the cursor to the end of a line. Press the Enter key.

## Editor Introduction

A blank line is created after the current line and the cursor moves to the first column position of this new line.

### Replacing and Inserting Text

The status area below the menu bar indicates which mode the editor is in. In *Replace* mode, the shape of the cursor is a solid block one, character width in size. Text under this block is replaced by any new text you type. In *Insert* mode, the shape of the cursor is a thin vertical line. Any new text you type is inserted into the file at the cursor position, and existing text right of the cursor shifts to the right.

To toggle back and forth between the two modes, press the Insert key. For example, place the cursor at about the middle of any line and do the following:

1. Check the cursor width to ensure that the Editor is in *Insert* mode. Type a few words. New text is inserted at the cursor position without overwriting existing text.
2. Press the Insert key to put the editor in *Replace* mode and type a few words. New text overwrites existing text at the cursor position.

### Splitting and Joining Lines of Text

To split a line so part of it forms a new line below the current line, do the following:

1. Move the cursor to where you want the line split. When you split a line, all text right of the cursor moves to a new line created below the current line.
2. Press Alt+S to split the line and leave the cursor at its current position, or press the Enter key to split the line and move the cursor to beginning of the new line.

To rejoin the lines, do one of the following:

- If the cursor is at the end of a line, press the Delete key to join the current line and the next line together.
- If the cursor is at the beginning of a line, press the Backspace key to join the current line and the previous line together.
- You can also place the cursor anywhere on a line and press Alt+J to join the current line and next line together.

**Deleting Text** You can delete unwanted text in many ways. Some are:

**Deleting single characters with the Delete key**

In *Insert* mode, the character right of the cursor is deleted. In *Replace* mode, the character under the cursor is deleted. In either mode, text right of the cursor moves left to fill the resulting gap.

**Deleting single characters with the Backspace key**

The character left of the cursor is deleted. Text right of or under the cursor moves left to fill the resulting gap.

**Deleting part of a line**

To delete all text between the cursor position and the end of the current line, press Ctrl+Delete.

**Deleting an entire line**

To delete a complete line, move the cursor to the unwanted line and press Ctrl+Backspace.

Before continuing, edit your file so it contains only the following lines:

```
/* This is only the very beginning! */  
/* And for now, this is the end. */
```

---

## Undoing Changes

The Editor records each set of changes you make to a file in an editing window. The number of changes made since the file was last saved is displayed on the status line below the Editor menu bar. If you want to undo a set of changes, do the following:

1. Select **Edit** from the Editor menu bar.
2. Select **Undo** from the resulting pull-down menu. The last set of changes to the file is undone.
3. Repeat the above steps until all unwanted changes are undone.

*Tip!* The accelerator key for Undo is Ctrl+Z.

### Exercise

Try this with the text you entered in the previous exercise:

1. Move the cursor to the left of the word *very* on the first line in your file.
2. Delete text from the cursor to the end of the line by pressing Ctrl+Delete.
3. Undo this change using the steps described above.
4. Press the backspace key a few times.
5. Undo the changes until the original line is restored.

## Editor Introduction

6. Type some text anywhere in the file. Move the cursor to the other line and type some more text.
7. Undo changes until the original lines are restored.

Before continuing with the next exercise, restore your file so it contains only the following lines:

```
/* This is only the very beginning! */  
/* And for now, this is the end. */
```

---

## Saving a File

To save your file, do the following:

1. Select the Editor window that contains the work you want to save.
2. Select **File** from the Editor menu bar.
3. Select **Save** from the resulting pull-down menu.
4. If you are saving an existing file, the file is saved under its current name. If you are saving a new file, the **Save as** window appears, as shown in Figure 30. Enter a new name for the file and click on the **OK** push button. The new file is saved under this name.

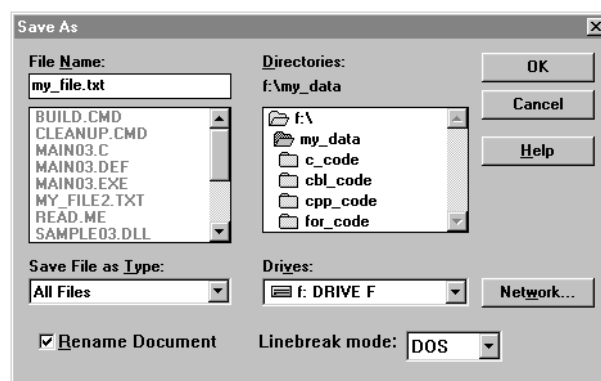


Figure 30. Save As Window

If you want to save changes to all files you are currently working with, do the following while in any Editor window:

1. Select **File** from the Editor menu bar.

2. Select **Save all** from the resulting pull-down menu.

Existing files are saved under their current file names. For each new file to be saved, the **Save as** window appears, as shown in Figure 30 on page 224. Enter the new name of the file and click on the **OK** push button to save the file under that name.

Before continuing with the next exercise, edit your file so it contains only the following lines. Save this file under the name `my_file.txt`.

```
/* This is only the very beginning! */  
/* And for now, this is the end. */
```

---

### Closing a Document View

The Editor lets you have more than one document or more than one view of a given document open at a time. This may affect the method you choose to close a document view.

- To close all documents and views of a document in the Editor, do the following:
  1. Select **File** from the Editor menu bar.
  2. Select **Exit** from the resulting pull-down menu. All Editor windows are closed. The Editor warns you if there are unsaved changes and gives you an opportunity to save them.
- To close only the current view of a document, do the following:
  1. Select **File** from the Editor menu bar.
  2. Select **Close view** from the resulting pull-down menu. The current view is closed. If this is the last view of a given document, the Editor warns you if there are unsaved changes and gives you an opportunity to save them.
- To close a ring, see “Working with Multiple Document Views” on page 239.

Close all documents and views before continuing with the next exercise.

---

### Opening an Existing File

To open an existing file, do one of the following:

- Drag a **File** icon from a **File Manager** window onto an **Editor** icon.
- If you are working in the WorkFrame environment, you can:
  - Double-click on an icon representing a source file, or,
  - Select an icon representing a source file, then select **Edit** from the pop-up menu for that icon. To raise an icon's pop-up menu, move your mouse pointer to the desired icon and press mouse button two.

## Editor Introduction

- At a command line prompt, type  
`lxpm filename.ext`  
where `filename.ext` is the name of the file you want to edit.
- If the Editor is already running, do the following:
  1. Select **File** from the Editor menu bar.
  2. Select **Open** from the resulting pull-down menu. The **Open file** window appears.
  3. Use the file and directory selection lists to select the file you want to open.
  4. Click on the **OK** push button to open the selected file for editing.

An Editor window opens showing the file you have chosen. You can begin editing this file, as described in “Entering and Editing Text” on page 220.

For this and the following exercises, open the `my_file.txt` file that you saved in “Saving a File” on page 224.

---

## Finding Text

To search for an item in your document, do the following:

1. Select **Edit** from the Editor menu bar.
2. Select **Find and replace...** from the resulting pull-down menu. A window similar to that shown in Figure 31 appears.

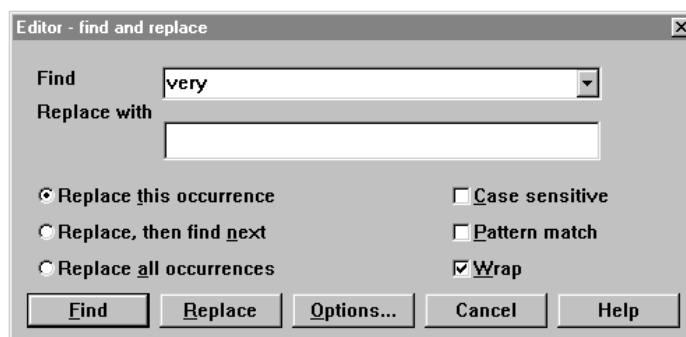


Figure 31. Finding Text

3. Type a search item in the **Find** text entry area. This can be a word, a partial word, or combination of such. You can also enter a pattern you want to match, provided that the pattern follows the rules of regular expression.



4. Select your desired options. If you enter a pattern in the **Find** text entry area, you must also select the **Pattern match** check box.
5. Click on the **Find** push button. The cursor moves to the next or previous occurrence of the search item, according to your chosen search direction.

**Exercise** Try this with the `my_file.txt` file:

1. Move the cursor to the beginning of the first line.
2. Find the word `very`.

---

### Finding and Replacing Text

To search for words and replace them with other words of your choice, do the following:

1. Select **Edit** from the Editor menu bar.
2. Select **Find and replace...** from the resulting pull-down menu.
3. Type the word you want to find in the **Find** text entry field.
4. Type the replacement word in the **Replace with** text entry field. Figure 32 shows an example.

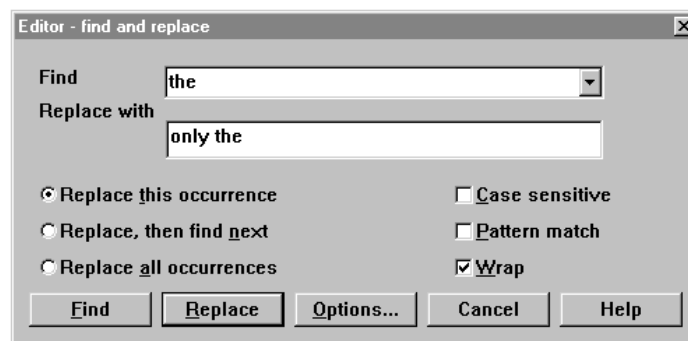


Figure 32. Finding and Replacing Text

5. Select your desired options.
6. Click on the **Replace** push button to find and replace your chosen word according to your selected options.

**Exercise** Try this with the `my_file.txt` file:

1. Bring up the **Find** window. Fill in the entries required to replace the word `the` with `only the`.

## Editor Introduction

2. Select the **Replace all occurrences** option.
3. Click on the **Replace** push button to change all occurrences of *the* to *only the*.
4. Look at the resulting file to see the changes.
5. Bring up the **Find** window. Fill in the entries required to replace the words *only the* back to the word *the*.
6. Select the **Replace, then find next** option.
7. Click on the **Options** push button. In the resulting **Find and replace - options** window, uncheck the **Cancel after find** check box, then click on the **OK** push button.
8. Click on the **Replace** push button. The first occurrence of *only the* is changed, then the cursor is moved to the next occurrence of *only the*. The **Editor - find and replace** window stays visible because of the option set in Step 7 above.
9. Continue clicking on the **Replace** push button until the message line at the bottom of the window indicates that there are no more occurrences of the word *only the* in the document. Then click on the **Cancel** push button.

---

## Finding a Specific Line in the File

A quick way to find a specific line in a file is to use the **Locate - line** window.

1. Select **Edit** from the Editor menu bar.
2. Select **Locate** from the resulting pull-down menu.
3. Select **Line...** from the cascaded menu. The **Locate - line** window appears, as shown in Figure 33. The **Line number** text entry field displays the line number where the cursor is now.

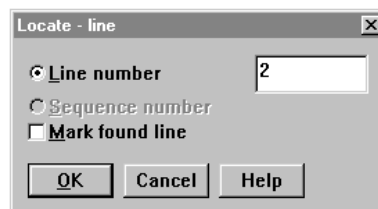


Figure 33. Locate Line Window

4. Replace that line number with the number of the line you want, and click on the **OK** push button. The **Locate - line** window disappears, and the cursor moves to the beginning of the chosen line. If you try to go to a line number that is greater

than the number of lines in the file, the cursor moves to the beginning of the last line in the file.

---

### Creating and Finding Marks

Marks function like bookmarks in a file. You can use marks to easily move around between *marked* positions in your file.

Marks have the following characteristics:

- Marks are not attached to a specific character, but to a specific cursor position. Changing text at a marked position does not alter the mark position.
- If lines are inserted or deleted before the line that a marked position is on, the mark shifts accordingly.
- If text is inserted or deleted to the left of a marked position, the mark shifts accordingly.
- If the marked position itself is deleted, the mark is lost.
- When you close a file, all marks are lost.

### Creating and Naming a Mark

To create and name a mark in your document, do the following:

1. Place the cursor at the position to be marked.
2. Select **Edit** from the Editor menu bar.
3. Select **Name a mark...** from the resulting pull-down menu. A window similar to that shown in Figure 34 appears.



Figure 34. Name a Mark Window

4. Enter the name of the new mark in the text entry field.
5. Click on the **OK** push button.

## Editor Introduction

### Finding a Named Mark

To move the cursor to a named mark, do the following:

1. Select **Edit** from the Editor menu bar.
2. Select **Locate** from the resulting pull-down menu.
3. Select **Mark...** from the next pull-down menu. A window similar to that shown in Figure 35 appears, listing all marks currently defined in the file.

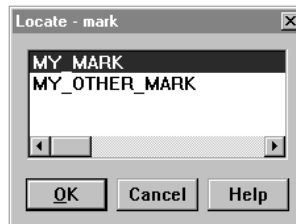


Figure 35. Find a Mark Window

4. Select the name of the mark you want to move to.
5. Click on the **OK** push button.

### Exercise

Try creating and finding a mark with the `my_file.txt` file:

1. Move the cursor anywhere on the first line of your file.
2. Create a mark called `my_mark`, as described in “Creating and Naming a Mark” on page 229.
3. Move the cursor anywhere near the middle of the second line.
4. Create a mark called `my_other_mark`.
5. Find the mark called `my_mark`, using the **Locate** choice from the **Edit** pull-down menu.
6. Move the cursor to the start of the second line, and insert some text. Then find the mark called `my_other_mark`.
7. Find the mark called `my_mark`. Press the Delete key to remove the marked position, then try to find the mark again.

## Editor Introduction

### Using a Quick Mark

The Editor offers a facility that lets you quickly mark a location in your document with a *quick mark*. You can have only one quick mark per document view. Setting a new quick mark replaces the previous quick mark, if one exists.

To set a quick mark, do the following:

1. Move the cursor to the position where you want to set a mark.
2. Select **Edit** from the Editor menu bar.
3. Select **Set quick mark** from the resulting pull-down menu. A quick mark is set at the chosen position in your file.

To find a quick mark, do the following:

1. Select **Edit** from the Editor menu bar.
2. Select **Locate** from the resulting pull-down menu.
3. Select **Quick mark** from the cascaded menu. The cursor is moved to the position marked by the quick mark.

### Exercise

Try this with the `my_file.txt` file:

1. Move the cursor to any position within your file.
2. Create a quick mark.
3. Move the cursor elsewhere in the file.
4. Find the quick mark using the method described above.

You can also find a quick mark using the procedure described in “Finding a Named Mark” on page 230. The quick mark appears in the list of named marks as `@QUICK`.

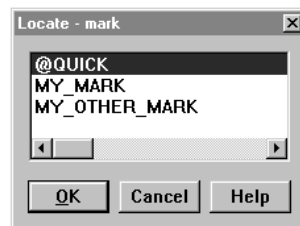


Figure 36. Find a Mark Window

## Editor Introduction

---

### Embedding Another File in the Current Document

To embed another file into your document, do the following:

1. Move the cursor anywhere on the line above where the inserted text is to appear.
2. Select **File** from the Editor menu bar.
3. Select **Get...** from the resulting pull-down menu. A window similar to that shown in Figure 37 appears.

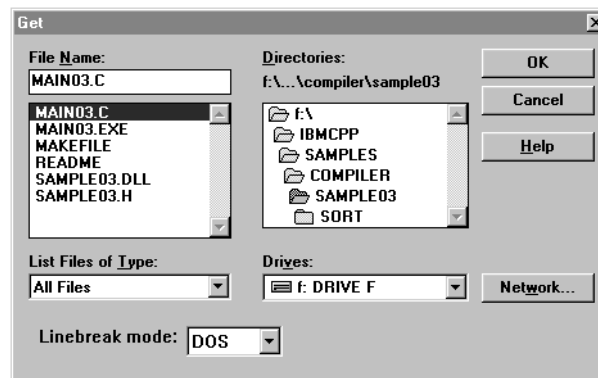


Figure 37. Get File Window

4. Select the desired file and click on the **OK** push button. The inserted text appears on the line(s) after the cursor position. The file from which text was received is unchanged.

### Exercise

Try this with the `my_file.txt` file:

1. Position the cursor anywhere on the first line in your file.
2. Insert the file from `\IBMCPP\SAMPLES\COMPILER\SAMPLE03\SAMPLE03.C` into your file.
3. Look at the resulting file, then undo the change.

---

### Issuing Editor Commands

The Editor is fully programmable. The following exercises describe how to issue commands to it:

1. Select **Actions** from the menu bar.
2. Select **Issue edit command...** from the resulting pull-down menu. The **Issue edit command** window appears. See Figure 38.

There are many commands you can use in the **Issue edit command** window. For a complete listing of editor commands, select **Help** from the Editor menu bar., then select **Editor reference** from the resulting pull-down menu. A list of reference topics appears. Select **Commands and Parameters** from this list, and any letter from the next list to see a list of commands and parameters starting with that letter. Click on any command or parameter item in the list to get more information on that item.

3. For example, in the text entry field, type:

bottom

and select the **OK** push button (or press Enter). The Editor performs the entered command, and the cursor moves to the last (bottom) character in the file. The **Issue edit command** window closes.

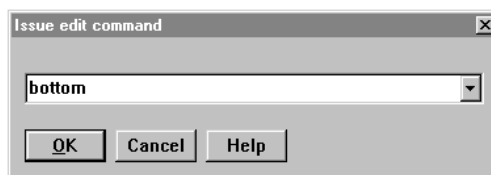


Figure 38. Issue Edit Command Window

4. Bring up the **Issue edit command** window again.
5. In the entry field, type:  
add 5  
Make sure to leave a space between the command and the number.
6. Click on the **OK** push button (or press Enter). Five lines are added at the end of the file after the cursor.

## Editor Introduction

### Issuing Multiple Editor Commands

The `mult` command lets you issue multiple commands using only one Editor command line entry. For example, to perform the two commands shown in “Issuing Editor Commands” on page 233 in one step, do the following:

1. Bring up the **Issue edit command** window and enter the following command:

```
mult ;bottom ;add 5
```

This example of the `mult` command includes two Editor commands: `bottom` moves the cursor to the last character in the file, and `add 5` adds five lines to the end of the file. Each Editor command is preceded by a `;`.

2. Click on the **OK** push button (or press Enter). Five new lines are added to the end of the file.

### More on Issuing Editor Commands

- To use a command again, click on the down arrow in the **Issue edit command** window to display previously used commands, and select one.
- You can use command synonyms when entering commands. For example, `;` is the editor default synonym for **mult** `;`. In step 1 above, you could have entered the command:

```
; bottom ;add 5
```

**Note:** If you choose to view the code in the profiles supplied with the editor, be aware that synonyms are used in these profiles.

- You can view a log of the messages produced by commands and macros you have used. Try this. Bring up the **Issue edit command** window and enter the following command:

```
query list.synonym
```

Then select **Windows** from the Editor menu bar, and **Macro log** from the resulting pull-down menu. The synonyms currently in use are listed at the bottom of the log.

- To get help for a command or determine what its return codes are, type *HELP command* in the **Issue edit command** window, where *command* is the name of the command for which you want information.



---

### Blocking and Manipulating Text

You can use the Editor block facilities to mark, or highlight, selections of text in your document. Marked text selections can then be manipulated in various ways.

#### Editor Block Manipulation Facilities

You can use standard operating system clipboard facilities to cut, copy, and paste selected text both within the editor and between the editor and other applications. The Editor also offers you other ways to quickly manipulate blocks of text within the editor.

**Default Editor Marking Mode** The Editor uses *stream marking* as its default marking mode. In this mode, the cursor is tied to the marked text selection. If you drag or move the cursor, the marked text selection is either changed or deselected. Also, if a block of text is marked, that text is replaced by the next character typed.

**Other Editor Marking Modes** You are probably familiar with stream marking, but if you are a programmer, the Editor has other marking modes that may be more useful for your work.

In these modes, the cursor is not coupled to the marked, or highlighted, text selection. You can mark a text block in a document or view, and copy that marked text block directly to the same or another document or view without using Windows clipboard facilities. Also, this lack of coupling means that typing within a marked text block does not cause the marked text block to be replaced. Instead, depending on whether the editor is in *insert* or *replace* mode, each typed character is either inserted before or replaces the character at the current cursor position.

You can change the default marking mode to one of the following choices:

- stream** Default. The cursor is coupled to the marked text selection.
- character** Similar to stream, except that the cursor is *not* coupled to the marked text selection.
- element** Whole elements, or lines are marked. The cursor is not coupled to the marked text selection.
- rectangle** Rectangular areas can be marked, starting at any character position on a line and extending over as many lines as needed. Rectangular blocking is useful for copying or deleting columns of text. The cursor is not coupled to the marked text selection.

## Editor Introduction

**Changing the Default Marking Mode** To change the default block marking mode, do the following:

1. Open the **Issue edit command** window by pressing Shift+F9.
2. Enter the following command,

```
set blockdefaulttype markmode
```

where markmode is one of the modes listed above.

## Unmarking a Block of Text

The following section shows you several ways to mark, or highlight, blocks of text. During and between exercises, you will also have to unmark these blocks.

To do so, use one of the following methods:

- Select **Edit** from the Editor menu bar, then select **Block**, and then select **Unmark**.
- Press Alt+U.
- Click mouse button two.

## Marking Blocks of Text

Some of the many ways to mark blocks of text are described below.

**Marking a Block of Text with the Mouse** To select a block of text with the mouse, do the following:

1. Position the pointer over the character that will start the block.
2. Press and hold mouse button one.
3. Drag the mouse pointer to the place where you want the block of text to end, and release the mouse button. The block is now marked, as shown in Figure 39 on page 237.

**Note:** The default block marking mode is used when marking text with the mouse.

**Exercise** Try this with the my\_file.txt file:

1. Mark the block only the very on the first line of your file.
2. Unmark the block using one of the methods described in “Unmarking a Block of Text.”

## Editor Introduction

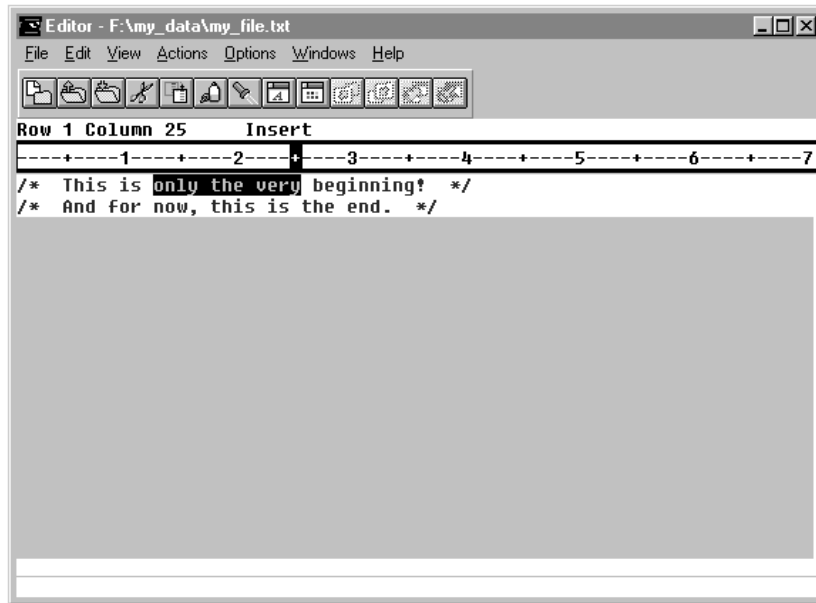


Figure 39. Marking a Block of Text

### Marking a Block of Text without the Mouse

You can mark a block of text without dragging a mouse by doing the following:

1. Use the mouse or the cursor keys to move the cursor to the character that will start the block, and press Alt+B.
2. Move the cursor to the place where you want the block of text to end, and press Alt+B again. The text between the two points is marked.

### Marking Complete Lines of Text

Place the cursor anywhere on a line and press Alt+L. The entire line is marked. To mark additional lines of text, move the cursor to the last line to be marked and press Alt+L. The selected lines and all lines between them are marked.

### Marking a Rectangular Block

To mark a section of text:

1. Move the cursor to one corner of the desired block, and press Alt+R.
2. Move the cursor to the opposite corner of the desired block, and press Alt+R again. The entire block between corner points is selected.

Alternatively, double-click mouse button two where you want the rectangular block to start, click and hold mouse button one, and drag to where you want the block to end. Release the mouse button to complete the marking of the block.

## Editor Introduction

### Manipulating Marked Blocks

Once you have marked a block of text, you can manipulate the marked block with either Windows standard clipboard-based facilities or the Editor's own block manipulation facilities. The following examples illustrate the Editor's facilities and the power of marking modes in which the cursor is not coupled to the marked text selection.

Rectangular blocks are used in the example. Set the Editor to perform rectangular blocking by doing the following:

1. Open a file of your choice for editing.
2. Bring up the **Issue edit command** window by pressing Shift+F9. Issue the following command to change the default marking mode:  
`set blockdefaulttype rectangle`

#### Copying a Marked Block

To copy a marked block, do the following:

1. Select and mark any rectangular block of text.
2. Move the cursor to the location in the file where the blocked text will be copied.
3. Select **Edit** from the Editor menu bar.
4. Select **Block** from the resulting pull-down menu.
5. Select **Copy** from the cascaded menu. A copy of the marked block is inserted at the cursor position.

Alternatively, position the cursor and then press Alt+C to copy a selected block.

#### Deleting a Marked Block

To delete a block that has already been marked, do the following:

1. Select **Edit** from the Editor menu bar.
2. Select **Block** from the resulting pull-down menu.
3. Select **Delete** from the cascaded menu to delete the marked block.

Alternatively, press Alt+D to delete a marked block.

#### Moving a Marked Block

To move a block that has already been marked to a different location in a file, do the following:

1. Move the cursor to the desired location.
2. Select **Edit** from the Editor menu bar.
3. Select **Block** from the resulting pull-down menu.
4. Select **Move** from the cascaded menu. The block is placed after the cursor.

## Editor Introduction

Alternatively, position the cursor and then press Alt+M, or use Alt+Z if you want to replace existing text.

### Exercise

Try this with your file:

1. Mark any word on the first line, and copy it to the end of the second line.
2. Mark a rectangular block of text. Move the cursor anywhere *outside* the marked block. Move the marked block to the new cursor location.
3. Delete the rectangular block that you have just moved.
4. Mark a complete line. Copy the marked line to a different line.

---

## Working with Multiple Document Views

The Editor lets you have several document views open for editing at the same time. These views can be of different documents, or different views of the same document.

Displaying multiple views of the same document helps you perform functions like cutting and pasting from one section to another. Any changes you make to one view are automatically updated in all other views of that document. You can also perform functions like cutting and pasting between views of different documents.

Collections of document views are organized into document *rings*, or groups of documents that share an editing window. When you open or create a document, that document either joins an existing ring or creates its own ring, depending on how your ring options are set.

Each ring owns its own editing window. A ring may reside within its own Editor window, or it may share an existing Editor window with one or more other rings. When rings share an Editor window, that Editor window is split either vertically or horizontally into multiple panes, with each pane containing the editing window for one ring.

Although you can have several editing windows *open* at any given time, only one editing window is *active* at a time. Except for global Editor commands, actions on the Editor menu bar apply only to the active editing window.

### Setting Default Ring Options

Ring options that you can set to help you manage rings in the Editor are:

**Default to ring** When this option is selected, a newly-opened document or view of a document is added to the current ring and appears in the current editing window. When not selected, each newly-opened document or view of a document is placed in its own ring, and a new Editor window for the ring appears.

## Editor Introduction

**Ring selector** When this option is selected, the editing window of each ring containing more than one document view includes a ring document-selector. See “Moving between Document Views within a Ring” on page 240 for more information about using the ring document-selector.

To set these options, do the following:


1. Select **View** from the Editor menu bar.
2. Select **Ring** from the resulting pull-down menu. The ring options appear as the last two entries in the cascaded menu. A check mark beside the option name indicates that the option is selected.
3. To change the selection state of a ring option, click on its cascaded menu entry. The option toggles to the opposite state.

### Moving between Document Views within a Ring

Within a document ring, only one view is accessible for editing at any given time. You can, however, cycle through all views in the ring by doing the following:

1. Select **Windows** from the menu bar of the Editor window that contains the ring you want to work with.
2. Select either **Next in ring** or **Previous in ring** from the resulting pull-down menu. The document visible in the editing window is replaced with the next or previous document in the ring.

If you have many document views in a ring, you may find it faster to go directly to another view by using the ring document selector. To use the ring document selector, do the following:

1. In the editing window containing the ring you want to work with, open the ring selector by clicking on the  push button found above the top right corner of the editing window. See Figure 40 on page 241 for an example.

## Editor Introduction

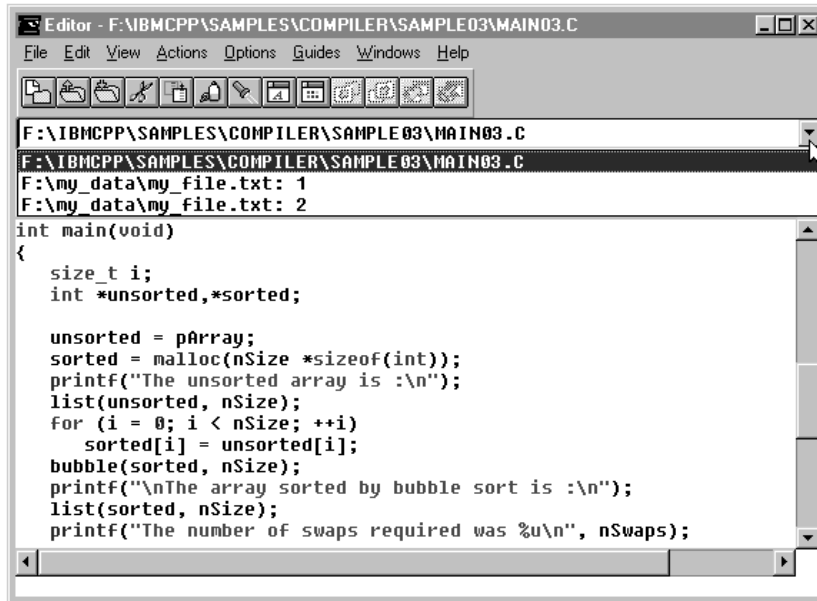


Figure 40. Ring Document-Selector

If you have more than one view open in your current ring and the ring document selector is not visible, you can enable it by doing the following:

- a. Select **View** from the menu bar of the Editor window that contains the ring you want to work with.
  - b. Select **Ring** from the resulting pull-down menu.
  - c. Select **Ring selector** from the cascaded menu to enable the ring document selector.
2. Click on the document view you want to work with. The selection list closes, and the view in the editing window is replaced with your newly chosen view.

### Creating Multiple Views of the Same Document

You can have more than one view of the same document open at a time, and these views may reside in more than one ring. Although you can scroll through or work on only one document view at a time, changes made to one view of a document are applied to all other views of that document.

To open another view of a document already open for editing, do the following:

1. Select the document for which you want to create another view.
2. Select **File** from the Editor menu bar.

## Editor Introduction

3. Select **Open new view** from the resulting pull-down menu. Depending on your ring option settings, a new view appears either in the editing window of your current ring, or in the editing window of a new ring.

Alternatively, you can:

1. Select the document for which you want to create another view.
2. Select **View** from the Editor menu bar.
3. Select **Split window** from the cascaded menu. The window containing the current view of the document is split to accommodate the editing window of a new ring. A new view of the current document is placed in this new ring. Figure 41 shows an example of two views of the same document appearing in a split Editor window.

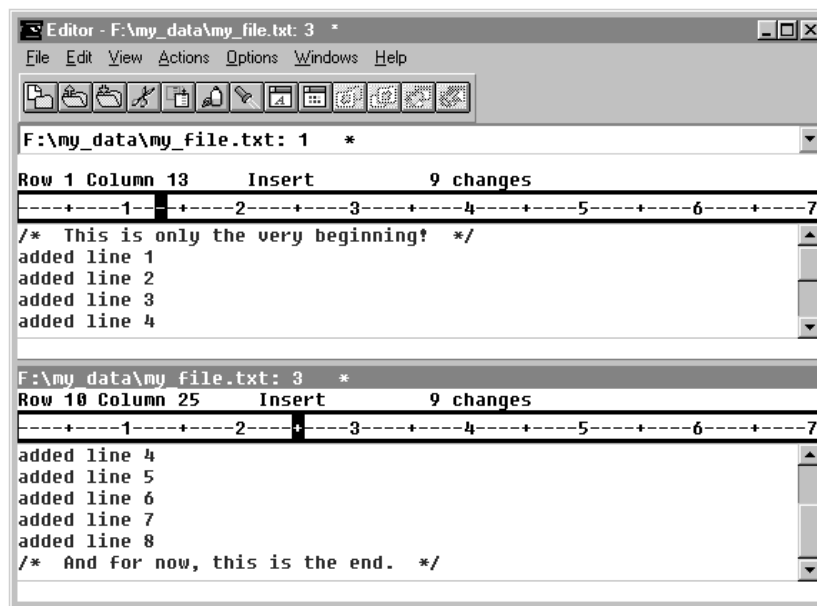


Figure 41. Split Editor Window with Two Views of the Same Document



### Moving a View to Another Ring

You may find it useful to move a document view to a new or other existing ring.

- To move a document view to a new ring, you can:
  1. Select the view you want to have moved to a new ring.
  2. Select **View** from the menu bar of the Editor window that contains the ring you want to work with.
  3. Select **Ring** from the resulting pull-down menu.
  4. Select **New ring** from the next pull-down menu. A separate Editor window for the new ring appears, and your document view is moved into the Editor window for the new ring.

Alternatively, you can select **Split ring** from this pull-down menu. The document is moved into a new ring. The Editor window that contained the original ring and editing window of the moved document is split to also accommodate the editing window of the new ring. The moved document appears in this new editing window.

- To move a document view to an existing ring, you can:
  1. Select **View** from the menu bar of any Editor window.
  2. Select **Ring** from the resulting pull-down menu.
  3. Select **Configure rings...** from the cascaded menu. The **Configure rings** window appears, as shown in Figure 42.

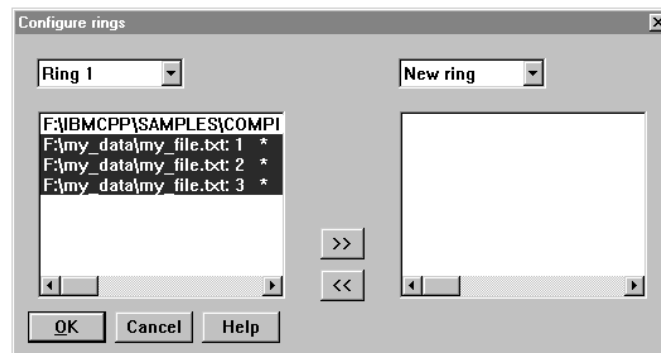




Figure 42. Configure Rings Window

4. Select your source and destination rings with the ring selection lists at the top of the **Configure rings** window.
5. Select one or more document views from one of the document view lists.

## Editor Introduction

6. Click on either the  or  push button to move the selected document views to your chosen ring.
7. Click on the **OK** push button when you have finished moving document views to other rings.

### Moving between Rings

If you have more than one ring open, you can move between rings by doing the following:

1. Select **Windows** from the menu bar of any Editor window.
2. Select **Next ring** or **Previous ring** from the resulting pull-down menu. The editing window belonging to the next or previous ring, depending on your choice, becomes the active window.

### Moving All Document Views into One Ring

To move all document views back into one ring, do the following:

1. Select **View** from the menu bar of the Editor window that contains the ring you want to work with.
2. Select **Ring** from the resulting pull-down menu.
3. Select **Single window** from the cascaded menu. All document views are moved to one ring, and all other rings are destroyed.

### Moving All Documents into Separate Rings

You can also quickly move each document into its own separate ring by doing the following:

1. Select **View** from the Editor menu bar.
2. Select **Ring** from the resulting pull-down menu.
3. Select **All new rings** from the cascaded menu. Each document view is placed into its own ring, and an Editor window appears for each new ring.

### Closing a Ring

To close a ring and all document views in it, do the following:


1. Select **File** from the menu bar of the Editor window that contains the ring you want to close.
2. Select **Close ring** from the resulting pull-down menu. All document views in the ring are closed. If the last view of any document is being closed, the Editor warns you of unsaved changes and gives you an opportunity to save them.

### Exercise

Try this with the `my_file.txt` file:

1. Make sure the **Default to ring** option is not checked. Use the procedure described in “Setting Default Ring Options” on page 239.
2. Open a new view.

## Editor Introduction

- a. Select **File** from the Editor menu bar.
  - b. Select **Open new view** from the resulting pull-down menu. The new view is added to a new ring. A new Editor window appears containing the editing window for the new ring. A second view of the `my_file.txt` document appears in the editing window, showing exactly the same text as the first.
3. Change some text in the second `my_file.txt` document view.
4. Cycle between the two views of the document. Because the views are in different rings, do the following:
  - a. Select **Windows** from the menu bar.
  - b. Select either **Next ring** or **Previous ring** from the resulting pull-down menu. The editing window for the next or previous ring becomes the active window.
  - c. Carefully inspect the text in the editing window of each ring. You will see that changes to the `my_file.txt` document view in one ring are also visible in the `my_file.txt` document view in the other ring.
5. Select the **Default to ring** option, as described in “Setting Default Ring Options” on page 239.
6. Open another document. This document is placed in the editing window of the current ring. The editing window shows a ring document selector because there is now more than one document view in the ring.
7. View the other document in the ring by doing the following:
  - a. Select **Windows** from the menu bar.
  - b. Select either **Next in ring** or **Previous in ring** from the resulting pull-down menu. The next or previous document view in the ring appears in the editing window.
8. Use the ring document selector to select a document view.
  - a. Click on the  push button of the ring document selector.
  - b. Click on a document or view name to select that item.
9. Close first one ring, then the other.

## Editor Introduction

### Using a Parser

The Editor supplies parsers for several programming languages such as C, C++, COBOL, and REXX. An emphasis parser uses colors and fonts to highlight different items in the program, but maintains the original indentation of the program.

To see how the parser works, do the following:

1. Open the \IBMCP\SAMPLES\COMPILER\SAMPLE03\SAMPLE03.C file for editing.

When a file is opened, the editor uses the file name's extension to select the appropriate load macro. A load macro contains a line that specifies which parser to load. When you opened the SAMPLE03.C file, the C parser was automatically invoked. A parser and its load macro also:

- Set default fonts and colors
- Set up special actions for keys
- Color and emphasize the data being displayed
- Set up additions to the **View** menu that contain selections specific to the chosen parser

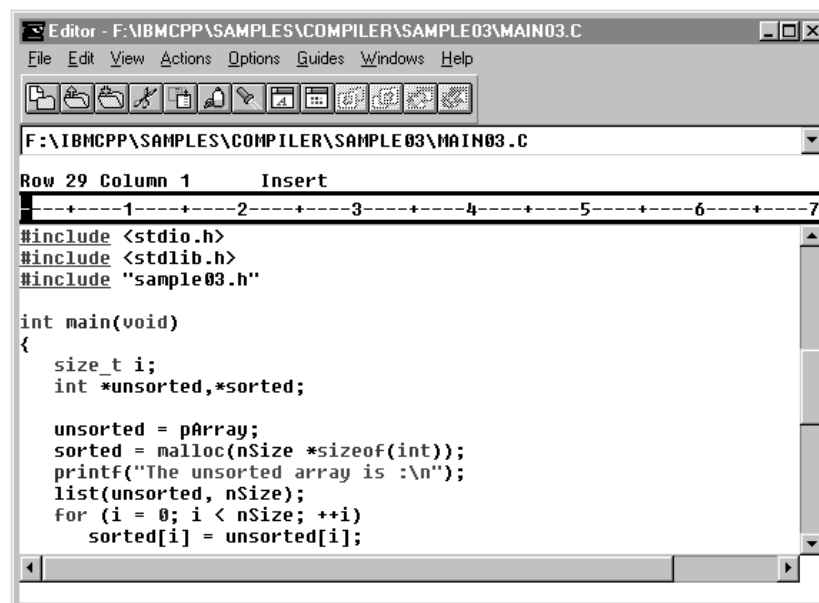


Figure 43. C-Language Program in Editor

2. Scroll the document to see the different colored fonts used to identify the different parts of a C language document.

### Displaying Different Parts of a Program

The Editor uses classes to describe the type of data that an element or line contains. Each element or line in a document may contain more than one class. A variety of colored fonts distinguish the different classes.

Classes defined in the C parser include:

<b>Comment</b>	Comments in the C code
<b>Code</b>	C language code
<b>Error</b>	Errors found by the C parser
<b>Function</b>	Function headers
<b>Brace</b>	Open or closed brace
<b>Semicolon</b>	Elements containing a semicolon statement terminator

To display different classes of the C-language document:

1. Select **Outline logic** from the **View** menu to display only the logic structure of the program sample. See Figure 44.

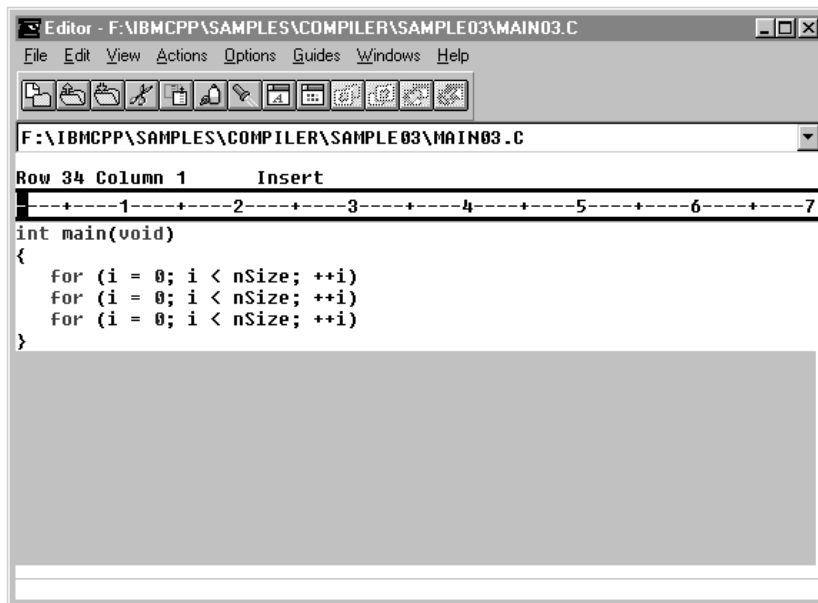


Figure 44. C-Language Program in Editor with Only Logic Structures Displayed

2. Select **Functions** from the **View** pull-down menu to display only the names of functions in the program sample.
3. Select **Include all** from the **View** pull-down menu to display all the lines.

## Editor Introduction

**Note:** Available **View** pull-down menu choices differ according to the type of file being edited.

The **View** pull-down menu choices are based on the **include** and **exclude** Editor commands. These commands let you view only elements that contain the specific classes that you choose. For example, to use the **include** command to display only the elements that contain items from the COMMENT class:

1. Select **Issue edit command...** from the **Actions** pull-down menu.
2. In the text entry box, type:  
`set include comment`
3. Click on the **OK** push button. Only lines with COMMENT class elements are displayed.
4. Select **Include all** from the **View** pull-down menu to display all classes again.

## Entering Some Code

The Editor is a live parsing editor because it monitors and records all the changes made to the document as you work. New data is colored or formatted as you add to its contents.

To type some lines of code into the sample C document:

1. Position the cursor in the program code, and press Enter several times to insert some blank lines.
2. Type in your own program code. When you have completed a line of code, either press the Enter key or click mouse button one. The line of code is parsed by the C parser.
3. Do not save changes to the sample file.



## Chapter 15. Customizing the Editor

You can customize the Editor to suit your preferences, redesigning anything from key assignments to more complex functions such as external commands and parsers.

This section covers some simple modifications you can perform. More complicated procedures require an extensive knowledge of the Editor commands and programming.

---

### Using the Editor Tool Bar

The Editor has an optional tool bar that you can use to invoke commonly used commands with a single mouse click. You can set the toolbar to display its items in different ways, or you can turn it off altogether. You can also define and add your own items to the tool bar.

To invoke an action from the tool bar, click on the appropriate tool bar item. As you move the mouse pointer over a tool bar item, a help balloon containing a description of that item's action appears near the tool bar item.

### Changing Tool Bar Display Characteristics

You can change the display characteristics of the tool bar by doing the following:

1. Select **View** from the Editor menu bar.
2. Select a tool bar type from the choices in the resulting pull-down menu. Choices available are:
  - Off (Tool Bar is not displayed)
  - Bitmap
  - Text
  - Bitmap and text

For example, if you select **Bitmap**, the Tool Bar appears as shown in Figure 45 on page 250.

## Customizing the Editor

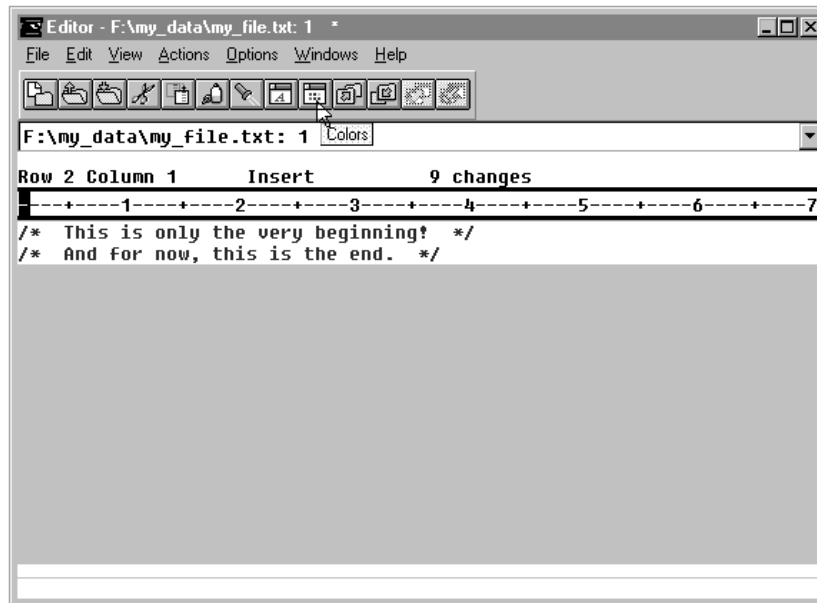


Figure 45. Editing Window with Tool-Bar Enabled

### Adding Your Own Items to the Tool Bar

You can add your own items to the tool bar. The following exercise shows how to add a simple text button called `my_button` to the tool bar to the right of the third item already in the tool bar. Pressing the button invokes the `mult ;bottom ;add 5` command, which adds 5 lines to the bottom of the file in the current editing window.

1. Set the tool bar to display **Text** or **Bitmap and text**, as described in “Using the Editor Tool Bar” on page 249.
2. Press Shift+F9 to open the **Issue edit command** window.
3. Enter the following command to define your new tool bar item. Press Enter or click on the **OK** push button when you are done.

```
set toolbar.my_button 3 mult ;bottom ;add 5
```

You can also define a contextual help balloon description for your new tool bar item. The following example expands the command shown above to define a short help balloon description:

```
set toolbar.my_button HELP "Add 5 line to file end" 3 mult ;bottom ;add 5
```

A new entry called `my_button` appears on the tool bar. If you place your mouse pointer over this new button, the button's help balloon description appears near your



## Customizing the Editor

new tool bar item. Clicking on the new button runs the `mult ;bottom ;add 5` command, which adds 5 lines to the end of your file.

You can also add icons to the tool bar. For more information, refer to *toolbar Parameter* in the Editor reference.

---

### Adding Your Own Items to the Menu Bar

You may want to customize the menus on the menu bar rather than create action keys because:

- Menus are visible and accessible immediately to the new user.
- Some menu items can be grayed out and disabled under certain conditions.
- Menus can be selected with the mouse.
- If desired, a mnemonic key can be assigned to a menu selection.
- If desired, an accelerator key can be assigned to a menu selection.
- You can create context-sensitive help for menus.

To create a new menu and to assign key sequences to each of its selections, use the **set actionbar** command as explained below. You can use the **set actionbar** command to add, modify, and delete menus on the Editor menu bar.

The menus and key sequences you create are lost when you close the file. To make them available each time you start the editor, see “Modifying Editor Behavior Permanently” on page 258.

#### Exercise

This exercise creates a new menu bar item called **Queries**. Pull-down menu choices associated with this item query various editor parameters. You can substitute the command of your choice. Some of the newly created pull-down menu items can be accessed by mnemonic and/or accelerator keys.

1. Press Shift+F9 to open the **Issue edit command** window.
2. Enter the following command to define the menu bar item and its first pull-down menu choice.

```
set actionbar.~Queries.Class 5 query class
```

The tilde (~) before the Q designates the Q as being the mnemonic for the menu bar selection. There is no mnemonic or accelerator key associated with the **Class** pull-down menu choice. The new menu bar item is placed to the right of the fifth item already on the menu bar.

## Customizing the Editor

3. Enter the following command to define another pull-down menu choice.

```
set actionbar.~Queries.F~onts\tCtrl+Z 5 query fonts
```

The new pull-down menu choice uses the letter o as a mnemonic, and displays Ctrl+Z as an accelerator key. To define the accelerator key action, you must use the **set action** command. For example:

```
set action.c-Z query fonts
```

4. Define one more pull-down menu choice.

```
set actionbar.~Queries.~Block_default 5 query blockdefaulttype
```

5. Test your customization by selecting the new menu bar item and its pull-down menu choices using the mouse, mnemonic key, and accelerator key.

---

## Customizing the Keyboard

The Editor provides two mechanisms for customizing the keyboard:

- A **set key** command lets you remap the connections between the physical keyboard and the logical keyboard. This remapping is global and affects all files.
- A **set action** command lets you program the logical keyboard, and assign a command or set of commands to a key. This programming is file related, so the same key can have a different effect for different files.

### Remapping the Keyboard Using the Set Key Command

Each physical key on the keyboard is mapped to a key on a logical keyboard. By default, each physical key is mapped onto the corresponding logical key, but this mapping can be altered with the **set key** command.

You can use **set key** to use the Editor with a restricted keyboard that does not have all of the required keys. For example, if your keyboard does not have the F11 and F12 keys, you can remap the Alt+1 and Alt+2 key combinations to have the same effect. Key in the following commands:

```
set key.A-1 F11
set key.A-2 F12
```

Similarly, you can use various accented characters that are not on the keyboard by assigning them to keys. The following example assigns lowercase and uppercase umlauts to Alt+A and Ctrl+A key combinations, respectively. The lowercase umlaut is represented by ASCII code 132, and the uppercase umlaut is represented by ASCII code 142.

```
set key.A-A 132      lowercase a umlaut
set key.C-A 142      uppercase A umlaut
```

## Customizing the Editor

**Note:** Ctrl and Alt key combinations do not recognize the difference between uppercase and lowercase characters. For example, using Alt+a or Alt+A in any Editor command yields the same result.

The **set key** command is sometimes more useful than a **set action** command because remapping using **set key** is global. For example, you could define the Alt+H key as the help key in all files by using the following Editor command:

```
set key.A-H F1
```

### Remapping the Keyboard Using the Set Action Command

When a key is pressed, the Editor performs an action dictated by the corresponding logical key. Some keys have default actions defined by the operating system, while others are defined by the Editor. Any key can be assigned an action using the **set action** command. For example, the following command makes the F7 key scroll the text in a window up one screen:

```
set action.F7 scroll screen up
```

The operating system reserves some keys, such as F1 and F10.

---

### Customizing the Autosave Facility for the Editor

The autosave facility saves a file automatically after a specified length of time has elapsed *or* a specified number of changes have occurred. You may want to customize the autosave facility for each file.

An autosave *triggered by time* occurs when a specified number of minutes has elapsed and at least one change has occurred during the elapsed time. The number of minutes elapsed must be greater than or equal to the duration set by the **autotime** parameter.

An autosave *triggered by changes* occurs when a specified number of changes have occurred. The minimum number of changes is specified by the **autocount** parameter, while the **autochanges** parameter keeps track of the number of changes made. When the total number of changes as recorded by **autochanges** is greater than or equal to **autocount**, an autosave occurs.

In both cases, the keyboard has to be idle for a set number of seconds, determined by the **idletime** parameter. This ensures that you are not typing data while the Editor is trying to autosave.

## Customizing the Editor

### Example

If you open a file with the following autosave variables:

- autotime 5
- autocount 50
- idletime 2

this means an autosave is triggered every 5 minutes, provided you have made at least one change. An autosave is also triggered after every 50 changes made to the file, regardless of the elapsed time. However, no autosave actually takes place unless you stop typing data for at least two seconds.

### Procedure

To customize the autosave settings for a file:

1. Use the **query** command to see the current autosave settings, for example:  
`query autotime`
2. Use the **set** command to change the autosave parameters, for example:  
`set autotime 15`

You can also use dialog windows to query and change many Editor parameters. For example, to query and change the **autocount** parameter, you could do the following:

1. Select **Options** from the Editor menu bar.
2. Select **Editor parameters...** from the resulting pull-down menu. The **Editor parameters** window appears.

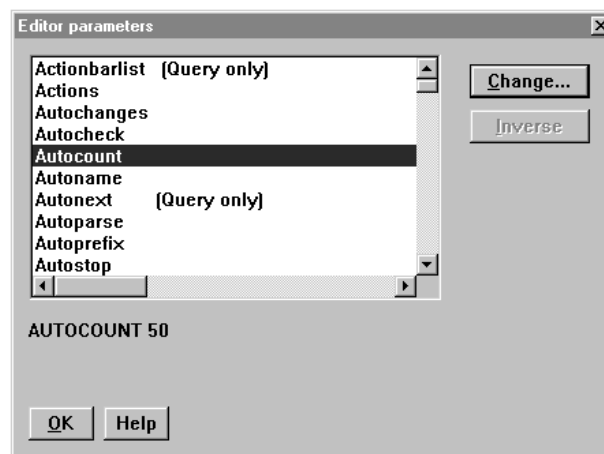


Figure 46. Editor Parameters Window

## Customizing the Editor

3. Find and highlight the **Autocount** entry in the parameters selection list. The current value for the **Autocount** parameter is displayed below the selection list. For example, in Figure 46, the current **Autocount** parameter of 50 is displayed.

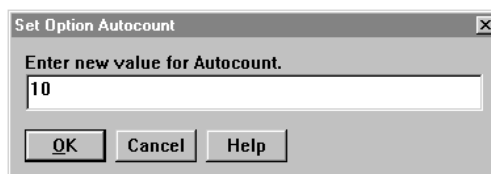


Figure 47. Set Option Autocount Window

4. Click on the **Change...** push button. The **Set Option Autocount** window appears.
5. Enter your new autocount into the **Set Option Autocount** window. For example, if you want an autosave after every 10 changes, enter 10 as shown in Figure 47.
6. Click on the **OK** push button to save the new **Autocount** parameter setting and close the **Set Option Autocount** window.
7. Click on the **OK** push button to close the **Editor parameters** window.

---

## Changing Editor Tab Settings

You can easily change the Editor tab stops to suit the requirements of the document you are working on. To change the tab stops, do the following:

1. Select **Options** from the Editor menu bar.
2. Select **Editor parameters...** from the resulting pull-down menu. The **Editor parameters** window appears.
3. Find and highlight the **Tabs** entry in the parameters selection list. The current value for the **Tabs** parameter is displayed below the selection list. For example, in Figure 48 on page 256, TABS EVERY 8 indicates that a tab stop occurs at every eighth character position on a line.
4. Click on the **Change...** push button. The **Set Option Tabs** window appears, as shown in Figure 49 on page 256.

## Customizing the Editor

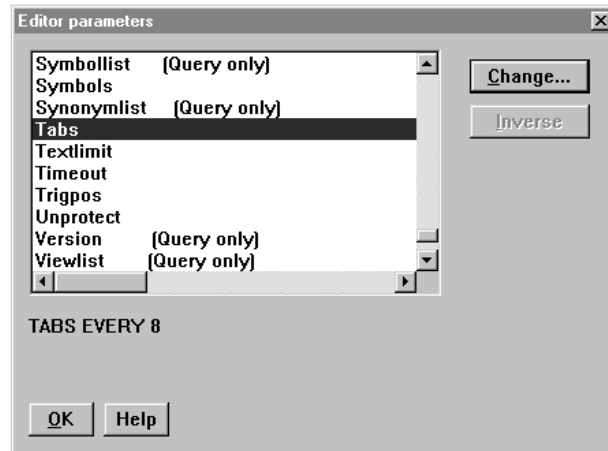


Figure 48. Editor Parameters Window

5. Enter your new tab stops into the **Set Option Tabs** window. For example, if you want a tab stop at every third position, enter EVERY 3.

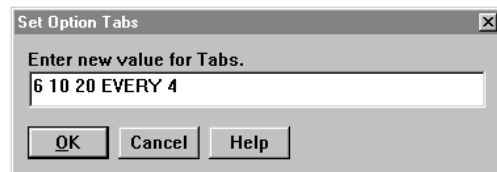


Figure 49. Set Option Tabs Window

You can also enter tab stops at irregular intervals. Figure 49 shows how you would set the **Tabs** parameter to have tab stops at character positions 6, 10, 20, and every fourth character position after position 20.

6. Click on the **OK** push button to save the new **Tabs** parameter setting and close the **Set Option Tabs** window.
7. Click on the **OK** push button to close the **Editor parameters** window.

---

## Changing the Base Editor Font

To customize the base character font used by the Editor, do the following:

1. Select **Options** from the Editor menu bar.
2. Select **Fonts...** from the resulting pull-down menu. A window similar to that shown in Figure 50 on page 257 appears.

## Customizing the Editor

3. Use the **Name**, **Size**, and **Font Style** selection lists to choose a base Editor font. The **Sample** window shows an example of the font currently chosen.
4. When you are satisfied with your chosen font, click on the **OK** push button. The chosen font is applied to the current Editor window and to all future edit sessions. To cancel all changes, click on the **Cancel** push button instead.

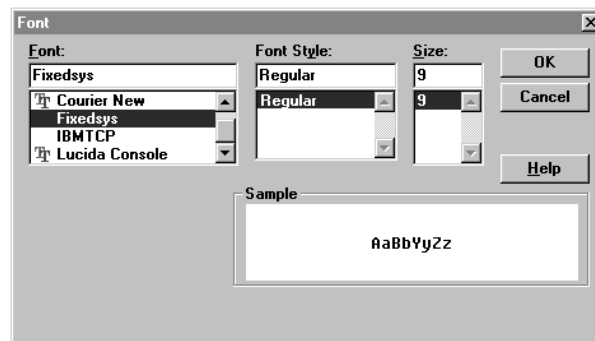


Figure 50. Font Window

---

## Changing Token Display Attributes

The Editor uses different font and character effects to emphasize different parts of a program. To customize the attributes assigned to different types of document elements, do the following:

1. Select **Options** from the Editor menu bar.
2. Select **Token attributes...** from the resulting pull-down menu. The **Token attributes** window appears as shown in Figure 51 on page 258.

The example in Figure 51 on page 258 shows that the foreground color coding for a C library function token is Bright Blue, with no special character emphasis.

3. Change token display attributes by doing the following:
  - a. Select the token type whose attributes you want to change from the **Token type** selection list. You can access this list by clicking on the down-arrow box at the right of the **Token type** entry field.
  - b. Select the **Foreground** radio button if you want to change the foreground color; otherwise, select the **Background** radio button.

**Note:** You can easily change the background color of all token types to the same color. Click on the **Change all backgrounds** push button after

## Customizing the Editor

selecting a background color for any token type. The background color chosen for that token type is applied to all other token types.

- c. Select any combination of character color and emphasis for the token type.
4. Repeat the above steps to change the display attributes of other token types you want to alter.

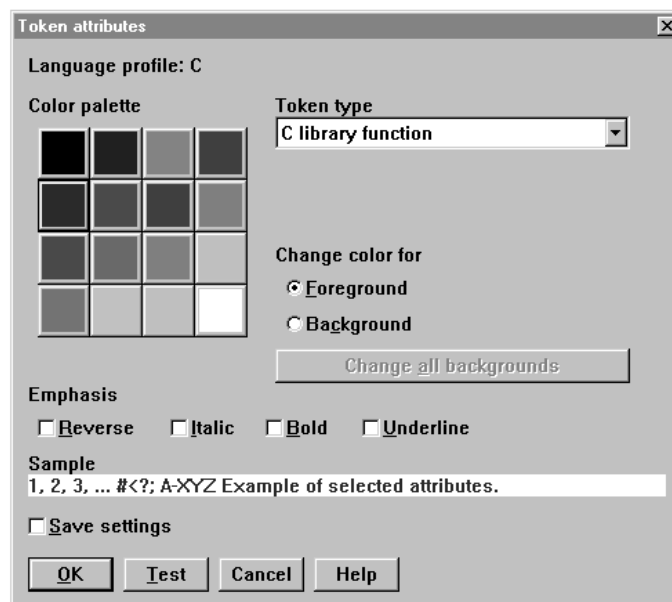


Figure 51. Token Attributes Window

5. If you want the new display attribute settings saved for use in future editing sessions, enable the **Save settings** check box. Otherwise, the new settings stay in effect only for the current editing session and are lost when you close the editor.
6. When you have finished setting token attributes, click on the **OK** push button.

---

## Modifying Editor Behavior Permanently

Most customizations remain in effect only for the current edit session. To use your customizations in future edit sessions, you must save them to a profile.

### Standard Editor Profiles

Much of the Editor's flexibility comes from *profiles*, which are text files containing editor commands. Some of the profiles supplied with the editor provide specific editing features, and run automatically at specified times.



## Customizing the Editor

The standard Editor profiles are kept in the \IBMCP\MACROS directory. You can edit these profiles directly, but **always** keep a backup. If you change them incorrectly, the behavior of the editor may be affected.

The following types of profiles are provided:

xxx.LXS	This profile runs whenever a file with filename type <i>xxx</i> is saved, including autosaves. Use of this profile is optional. For example, you could create a CPP.LXS profile to be run when files with a CPP source type are saved.
xxx.LXL	This is a load profile containing commands (such as parser selection) specific to files with a certain source type. For example, the C.LXL load profile is run if a file with a filename extension of C is opened.
PROFILE.LX	This is a profile in which you can save your own customization commands. It is the last standard profile that is run before a file is opened, and its commands override the commands used in previous profiles.

### User-Defined Load Profiles

You can directly modify the IBM-supplied Editor load profiles to your own personal preferences, but your modifications may be lost if you later reinstall the editor. To avoid losing your load profile modifications:

1. Do not directly modify the *xxx.LXL* standard load profiles.

Instead, create an *xxx.LXU* user load profile in the \IBMCP\MACROS directory, where *xxx* is the file type to which this profile applies.

For example, if your personalized load profile applies to .C files, your personalized load macro must be called C.LXU.

2. In the *xxx.LXU* load macro, include only the editor commands needed to modify the standard load profile behavior to your preferences.
3. When you load the Editor with file type *xxx*, the Editor loads your *xxx.LXU* user load profile immediately after the *xxx.LXL* standard load profile.

### Storing Personalized Profiles

Personalized profiles should be stored in your own directory. This helps ensure that your profiles are not lost if you later reinstall the Editor. It also simplifies profile management when the editor is used in a network environment.

To store your profiles in your own directory, do the following:

1. Create a directory in which to store your personal profiles, such as:  
`mkdir \my_prof`

## Customizing the Editor

2. Create your personalized profiles and store them in this directory.
3. Add the new directory to the LPATH environment variable in your config.sys file.  
If this environment variable does not already exist, create it. For example:  

```
set LPATH=d:\my_prof;d:\lpex\macros;
```
4. Reboot your system.

### Sample Personalized Profile

This exercise saves some of the customizations performed in this chapter to the PROFILE.LX file.

Perform the following steps:

1. Select **Options** from the menu bar.
2. Select **Profiles** from the resulting pull-down menu.
3. Select **User preferences** from the next menu. The Editor loads the PROFILE.LX file for editing.

**Note:** To open all active profiles for editing, select **All active** from the menu. Be sure you have made backup copies of the standard Editor profiles before you try to edit them.

4. Modify the PROFILE.LX file to contain the following lines. The first line of the profile must be a comment, as shown. Do not forget to put quotation marks around each command line.

```
/* profile.lx */  
'set blockdefaulttype rectangle'  
'set toolbar.my_button HELP "Add 5 line to file end" 3 mult ;bottom ;add 5'  
'set actionbar.^Queries.Class 5 query class'  
'set actionbar.^Queries.F^onts\tCtrl+Z 5 query fonts'  
'set action.c-Z query fonts'
```

5. Save the file and quit the Editor. Your new Editor customizations take effect the next time you start the Editor.
6. To cancel your customizations, edit the PROFILE.LX file to remove the unwanted Editor commands. Save the modified file.

**Note:** If you later reinstall the Editor software, you will lose your customizations. See “Storing Personalized Profiles” on page 259 for information on how to safely store your personalized profiles.

---

## Part 5. Debugging Your Program

This part of the *User's Guide* describes the VisualAge for C++ Debugger, which you can use to debug your programs once you have compiled and linked them.

---

<b>Chapter 16. Introduction</b>	263
Hardware Requirements	263
Software Requirements	263
Compiling and Linking Your Program	263
<b>Chapter 17. Getting Started</b>	265
Debugging Win32 Applications on Windows NT or Windows 95	265
Debugging Win32s Applications	266
Using the Process List Window	268
Starting the Debugger from WorkFrame	269
Ending the Debugging Session	270
<b>Chapter 18. Frequently Used Features of the Debugger</b>	271
Understanding the Debug on Demand Feature	271
Using the Tool Bar	271
Executing a Program	273
Setting Breakpoints	274
<b>Chapter 19. Introducing the Main Debugging Windows</b>	275
Using the Debug Session Control Window	275
Using the Source Windows	293
<b>Chapter 20. Introducing the Other Debugging Windows</b>	305
Using the Call Stack Window	305
Using the Register Window	308
Using the Storage Window	309
Using the Local Variable Window	313
Using the Monitor Windows	316
Using the Breakpoint List Window	316
<b>Chapter 21. Expressions Supported</b>	323
Supported Expression Operands	323
Supported Expression Operators	324
Supported Data Types	325

---

## **Debugging Your Program**



## Chapter 16. Introduction

This section introduces the VisualAge for C++ Debugger and lists the hardware and software requirements. It also lists the compiling and linking options.

The VisualAge for C++ for Windows Debugger (hereafter referred to as debugger) is a source-level debugger which uses Windows graphical user interface. It is used for debugging 32-bit Windows applications that are running on Windows NT, Windows 95, and the Win32s subsystem on Windows 3.11. The debugger helps you detect and diagnose errors in code written in C and C++ languages.

With the debugger you can:

- Manage execution in applications and DLLs
- Control breakpoints
- Display and modify program states using storage, register, variables, and call stack windows
- Control threads in multi-thread environments
- Analyze a program just in time using the **Debug on demand** feature
- Remotely debug programs operating on a Win32s subsystem.

---

### Hardware Requirements

The debugger must be run on an Intel-based system capable of running Windows NT 3.51 and higher or Windows 95. TCP/IP is required to enable the communication services for remote debugging.

---

### Software Requirements

The debugger is installed as part of VisualAge for C++ product. To enable remote debugging, the debug probe needs to be installed on a Windows 3.11 machine. See “Debugging Win32s Applications” on page 266 for details.

---

### Compiling and Linking Your Program

Before using the debugger, compile and link your program with the following options:

- |      |                                                                                                                                           |
|------|-------------------------------------------------------------------------------------------------------------------------------------------|
| /Ti+ | Compiles your program to produce an object file that includes line number information and a symbol table, in addition to the source code. |
| /O-  | Compiles your program with optimization off. This is the default.                                                                         |

## Debugging Your Program

`/Oi-` Compiles your program with inlining off. This is the default.

`/DEbug` Links your program to produce an executable file that includes line number information and a symbol table, in addition to the executable code.

**Note:** When you specify the `/Ti+` option with the `/DEbug` option, `icc` passes this option to the linker automatically, so you only need to use it if you link separately from the compile.

## Chapter 17. Getting Started

This section describes how to start a debugging session from either a Windows NT or Windows 95 command prompt and how to end the session. It also describes how to debug Win32s applications and introduces the **Process List** window.

### Debugging Win32 Applications on Windows NT or Windows 95

To start the debugger from either a Windows NT or Windows 95 command prompt, enter the command `idebug` and the following parameters in the order they are listed:

1. Any debugger parameters that you want to use.
2. Name of the program you want to debug.
3. Any input parameters that you want to pass to the program.

For example, type the following:



```
idebug /x myprog xyz
```

where `/x` represents a debugger parameter, *myprog* represents your program name, and *xyz* represents the program parameter you want to pass to the program.

The debugger parameters are:

- `/p+` Use program profile information.
- `/p-` Do not use any program profile information.
- `/i` Start the debugger in the system initialization routine so that you can debug initialization code.

If you type `idebug` and press Enter, the **Startup** window displays.

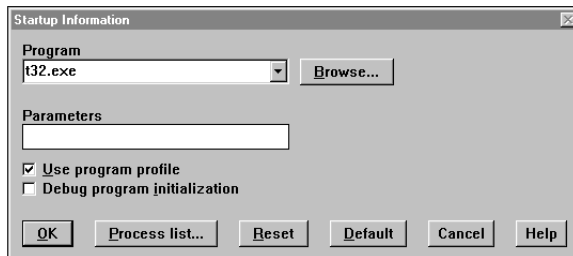


Figure 52. Startup Window

Use the **Startup** window to specify the program you want to debug.

## Getting Started

- Type the name of the program you want to debug in the **Program** entry field. You can also select the **Browse** push button.

If you select **Browse**, the **File** window displays. From this window, select the program you want and select **OK**. The **Startup** window displays again with the program name you selected displayed in the **Program** entry field.

You can also select the list button to display a drop-down combination box. The drop-down combination box contains a list of up to five previously debugged programs.

- In the **Parameters** field, type any parameters that you want to pass to your program. You must separate multiple parameters with spaces.
- Enable the **Debug program initialization** check box to start the debugger in the system initialization routine so that you can debug initialization code. This is optional.
- Enable the **Use program profile** check box to restore the debugger windows and breakpoints when debugging a program more than once. It is stored separately for each program debugged. This is optional.
- Select the **Process list** push button to display the **Process List** window. From this window, you can attach to processes that you want to debug. This is optional. See “Using the Process List Window” on page 268 for more information on the **Process List** window.
- Select the **OK** push button to start the debugging session.

---

## Debugging Win32s Applications

The debugger also debugs Win32s programs running on a Win32s system. The debugger and the debug probe communicate using TCP/IP. The user interacts with the debugger on Windows NT or Windows 95 (*client*). The debug probe (*server*) runs on the Win32s system and controls the execution of the debuggee. A copy of the binary and source code of the debuggee should be placed on a Windows NT or Windows 95 machine.

The following procedure illustrates how to set up your environments for Win32s debugging:

1. Start the debugging probe (*idebugp.exe*) on the Win32s system.
2. Note the host address and port id displayed in the probe monitor. You can also specify the port number as an argument by selecting the **Run** choice from the **File** menu.
3. On the Windows NT or Windows 95 command session where the debugger is executed, set up the following environmental variables:



## Getting Started

### **CAT\_MACHINE**

Identifies the server IP address and port number. Specify this as Remote\_IP\_Address:port. The port number is assigned during the server invocation in step 1 on page 266.

For example, type the following at the command prompt:

```
set cat_machine=192.100.104.2:6000
```

### **CAT\_PATH**

Identifies where the source for the debug binaries resides on the client machine. This is a path.

```
set cat_path=c:\source
```

### **CAT\_LOCAL\_PATH**

Identifies the directory on the client machine where the debug binaries resides.

For example, type the following at the command prompt:

```
set cat_local_path=path;
```

where *path* is the location of your debug binary.

### **CAT\_LAUNCH\_TYPE=WIN32S**

Identifies Win32s as the target platform.

### **CAT\_COMMUNICATION\_TYPE=TCP**

Identifies communication protocol. TCP/IP is the only protocol supported in this release.

### **CAT\_REMOTE\_PATH**

Identifies the directory on the server machine (Win32s system) where the debug binaries resides.

```
set cat_remote_path=C:\application
```

Trailing path delimiters ( \ ) are not required but are accepted.

### **CAT\_REMOTE\_OS=YES**

Required for remote debugging. You have to remove this environment variable for native debugging.

```
set cat_remote_os=yes
```

4. On the client machine, place the binary or binaries in the CAT\_LOCAL\_PATH directory. You also need to copy the source files to the CAT\_PATH directory.
5. To start the debugging session, type the following at the command prompt:

```
idebug executable_file_name
```

## Getting Started

### Understanding the Search Path

The search path tells the debugger where to find the source file used in the source windows. The debugger searches for the source files in the following order:

1. The path defined by the CAT\_LOCAL\_PATH environment variable.
2. The path where the executable file is located.
3. The path defined by the CAT\_PATH environment variable, if specified.
4. The current path.

To override the normal search order, use the CAT\_OVERRIDE environment variable. To set the CAT\_OVERRIDE environment variable, type the following at the command prompt:



Set CAT\_OVERRIDE=path

where, *path* is the location of your source files. If the source file is not found in the defined override path, the debugger uses the normal search order.

To set the CAT\_PATH environment variable, type the following at the command prompt:



Set CAT\_PATH=path

where, *path* is the location of your source files.

To set the number of spaces between tab stops in your source code, type the following at the command prompt:



Set CATTABGRID=*n*

where, *n* is the increment number of spaces between tab stops. For example, if *n* is 5, tab stops would be 5, 10, 15, and so on.

### Limitations

The following functions are not available when debugging Win32s applications:

- Halt a running application
- Load occurrence breakpoint.
- Run exception and step exception in the **Application Exception** window.

---

### Using the Process List Window

Use the **Process List** window to attach a process that you want to debug. You can only attach to one process at a time.

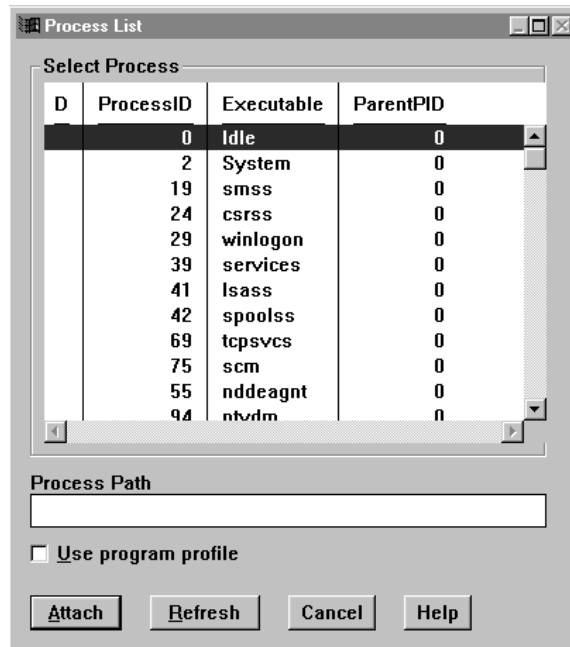


Figure 53. Process List Window

To select a process to debug:

1. Type the path name of the process executable in the **Process path** entry field.
2. Select the process you want to debug from the list.
3. Select the **Attach** push button.

You can also double-click on the process name to select it. However, you must still provide the path name in the **Process path** entry field.

**Note:** If you are currently debugging a process, the process will be terminated when a new process is attached. You can only debug one process at a time. Do not attach to operating system processes. This could cause unpredictable results.

You can also use the **Attach** feature to debug child processes.

**Note:** This feature is not available for Windows 95.


---

## Starting the Debugger from WorkFrame

Before you start the debugger from the WorkFrame environment, you must create a project for the program you want to debug. To compile and link a target program

## Getting Started

with debugging information, you must set the debugger options that the WorkFrame environment uses for creating a project.

 For information on creating a project, setting options and starting the debugger, refer to the WorkFrame documentation.

---

## Ending the Debugging Session

To end the debugging session, select **Close debugger** from the **File** menu in a debugger window. The **Close Debugger** window displays. Select one of the following choices:

- Select **Yes** to end your debugging session.
- Select **No** to return to the previous screen without exiting the debugger.

You can also end the debugging session by pressing F3 in any of the debugger windows.

**Note:** If you are debugging remotely and you press F3, the debugger will close but the application that is running on the remote machine will not.



## Chapter 18. Frequently Used Features of the Debugger

This section introduces the **Debug on demand** feature, the tool bar, ways to execute your program, and how to set breakpoints.

---

### Understanding the Debug on Demand Feature

**Debug on demand** enables the system to open a debugging session whenever an error occurs in your application. For example, if an unhandled exception occurs and **Debug on demand** is enabled, the debugger is notified that an error has occurred. The debugger starts and attaches to your application at the point of fault. This can save you time because you don't have to re-create errors. Furthermore, your application will run at full speed without any interference from the debugger until an exception is encountered. It is also useful when you are debugging asserts in place.

VisualAge for C++ can start **Debug on demand** for any application that fails while it is running, even if the application does not contain debug information. Using **Debug on demand**, it is possible to find and fix the problem in the application and let it continue running.

**Note:** This feature is only available on Windows NT.

To enable the **Debug on demand** feature, type the following at a command prompt:

```
dod path_name\idebug
```



where, *path\_name* is the location where the idebug.exe was installed and *idebug* is specified as the default debugger. For example, you would type:

```
dod e:\ibmcpp\bin\idebug
```

To disable the **Debug on demand** feature, type the following at a command prompt:

```
dod.exe /u
```

---

### Using the Tool Bar

A tool bar has been provided for easier access to frequently used features. The following is a list of features that are provided:



**Step over** executes the current line in the program. If the current line is a call, execution is halted when the call is completed.

## Frequently Used Features of the Debugger



**Step into** executes the current line in the program. If the current line is a call, execution is halted at the first statement in the called function.



**Step debug** executes the current line in the program. The debugger steps over any function for which debugging information is not available (for example, library and system routines), and steps into any function for which debugging information is available.




**Step return** automatically executes the lines of code up to, and including, the return statement of the current function.



**Run** allows you to start and stop the program.

When the debugger is running, the **Run** button is disabled

and the **Halt** button  is enabled. You can click on the **Halt** button to halt the program execution. You can also interrupt the program you are debugging by selecting the **Halt** choice from the **Run** menu.



**View** changes the current source window to one of the other source windows. For example, you can change from the **Disassembly** window to the **Mixed** window.



**Monitor Expression** displays the **Monitor Expression** window, which allows you to type in the name of the expression you want to monitor.



**Call Stack** displays the **Call Stack** window, which allows you to view all of the active functions for a particular thread including the system calls. The functions are displayed in the order that they were called.



**Registers** displays the **Registers** window, which allows you to view all the processor and coprocessor registers for a particular thread.



**Storage** displays the **Storage** window, which shows the storage contents and the address of the storage.

## Frequently Used Features of the Debugger



**Breakpoints** displays the **Breakpoints List** window, which allows you to view all the breakpoints that have been set.



**Debug Session Control** displays the **Debug Session Control** window.



**Growth direction** allows you to change the direction that items are displayed on the stack.



**Delete** allows you to delete the highlighted item.



**Delete all** allows you to delete all the items in the window.



**32-float** displays the storage contents as a 32-bit floating point.



**64-float** displays the storage contents as a 64-bit floating point.



**32-bit unsigned** displays the storage contents as a 32-bit unsigned integer.



**32-bit signed** displays the storage contents as a 32-bit signed integer.



**ASCII** displays the storage contents in ASCII.



**Hex and ASCII** displays the storage contents in Hex and ASCII.



**Change representation** allows you to change the data representation.

---

## Executing a Program

You can execute your program by using step commands or the **Run** command.

**Step commands** Step commands control the execution of the program. The execution of the line of code is reflected in all open views and is performed in the thread specific to the view.

## Frequently Used Features of the Debugger

The step commands are located in the tool bar of the source windows and under the **Run** menu of the source windows.

**Run command** The **Run** command runs the program until a breakpoint is encountered, the program is halted, or the program ends.

You can start the **Run** command from the **Run** button in tool bar or the **Run** menu of the source windows.

When you execute your program, a clock icon displays to indicate that the program is running and might require input to continue to the next breakpoint or termination of the program.

---

## Setting Breakpoints

You can control how your program executes by setting breakpoints. A breakpoint stops the execution of your program at a specific location or when a specific event occurs.

To set breakpoints, select the **Breakpoints** menu from the **Debug Session Control** window or from any of the source windows and select the appropriate choice for the type of breakpoint you want to set. You can also set a simple line breakpoint by double-clicking in the *prefix area* of an executable statement in any of the source windows. The prefix area is the area to the left of the source code where line numbers or addresses are displayed. The prefix area turns *red* indicating that the breakpoint has been set. You can also use the arrow keys to move the cursor to the prefix area and then use the space bar to toggle the breakpoints. This allows you to set or delete a breakpoint.

Refer to “Breakpoints Menu Choices” on page 279 for more information on breakpoints.



## Chapter 19. Introducing the Main Debugging Windows

This section introduces the **Debug Session Control** window and the three source windows which offer different views of your source code. It describes the menu items and choices that are located in each of these windows.

---

### Using the Debug Session Control Window

The **Debug Session Control** window is the control window of the debugger and displays during the entire debugging session. This window is divided in two panes. One pane shows the threads for the program you are debugging and the other shows the components for the program you are debugging.

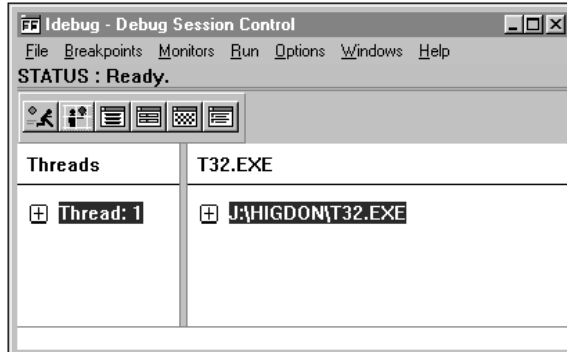


Figure 54. Debug Session Control Window

The **Threads** box contains the threads and the state of the threads started by your program.

To display the state of a thread, select the plus icon to the left of the thread. To enable or disable the thread listed in the **Threads** pane, double-click on the word **Enabled** or **Disabled** depending on the state of the thread. You can also toggle the **Thread enabled** choice from the **Run** menu. When a check mark symbol displays beside the **Thread enabled** choice, threads are enabled and the debugger allows the highlighted thread to execute. When the check mark symbol does not display, threads are disabled and do not execute.

The **Thread Popup Menu** contains choices to take you to different debugger windows. The popup menu is shown when you place the mouse pointer on a highlighted thread name and click mouse button 2.

## Introducing the Main Debugging Windows

The following are the popup choices:

<b>Execution point</b>	Opens a source window containing the next line to be run.
<b>Disable</b>	Disables a thread stopping it from being executed.
<b>Enable</b>	Enables a thread allowing it be executed.
<b>Registers</b>	Displays the <b>Register</b> window.
<b>Call stack</b>	Displays the <b>Call Stack</b> window.
<b>Local variables</b>	Displays the <b>Local Variables</b> window.

The **Components** box shows the path name of the executable file that is associated with the program you are debugging.

To display a list of object files contained within an executable file, select the plus icon to the left of the executable file name. To open a source window of an object file, double-click on the object file name.

To display a list of functions for a specific object file, select the plus icon to the left of the object file name. To open a source window of a specific function, double-click on the function name.

You can display any object or function in a source window by double-clicking on the name in the **Components** box or by highlighting the component name and selecting the **View** popup choice.

To specify which components are shown in the **Components** list, select **Options -> Window Settings -> Display style**. The **Display Style** window displays. Disable the **Show all components** choice if you want only components compiled and linked with debugging data to be listed. If the choice is enabled, all components are listed.

The **Object or Function Popup Menu** contains choices that allows you to display the object or function in a source window or set a function breakpoint. The popup menu is shown when you place the mouse pointer on a object or function name and click mouse button 2.

The choices in the popup menu are:

<b>View</b>	Shows the object or function in a source window.
<b>Set function breakpoint</b>	Sets a function breakpoint to stop the execution of your program after calling a specific function.
<b>Set breakpoint (every)</b>	Sets a breakpoint to stop the execution of your program on a highlighted variable or expression in all the threads.

## Introducing the Main Debugging Windows

### File Menu Choices

Select choices from the **File** menu of the **Debug Session Control** window to open a new source file, locate a particular function, open a source window that contains the next line to be executed, start a new debugging session, select a process to debug, or end a debugging session.

#### Open new source...

Displays the **Open New Source** window, which allows you to open a new source file. If you have multiple source files in your program, only one source file is initially displayed. Use the **Open New Source** window to open additional source files.

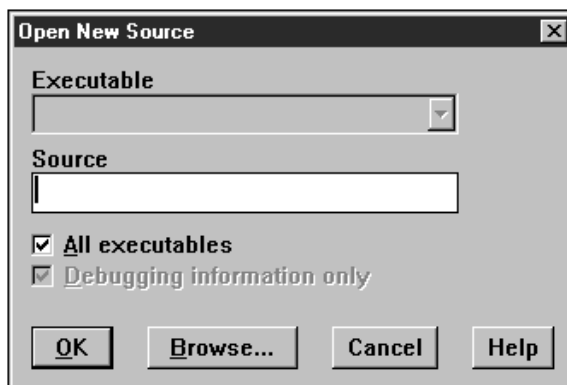


Figure 55. Open New Source Window

To use the **Open New Source** window:

- Type the name of the object file you want to open the source for in the **Source** entry field. For example, to look for the source used to compile A123.OBJ, type the following:

A123.



If you are unsure of the file name, select the **Browse** push button to view a list of the files that you can select.

- Type the name of the executable file in the **Executable** entry field. The source files for the executable file display in the **Source** entry field.
- Enable the **All executables** check box if you want to search all the executable files. Disable the **All executables** check box to search for a particular executable file.
- Enable the **Debugging information only** check box if you want to search only the source files that contain debugging information.
- Select the **OK** push button.

## Introducing the Main Debugging Windows

### Find function...

Displays the **Find Function** window, shown in Figure 56 on page 278, which allows you to open a source window to a particular function.

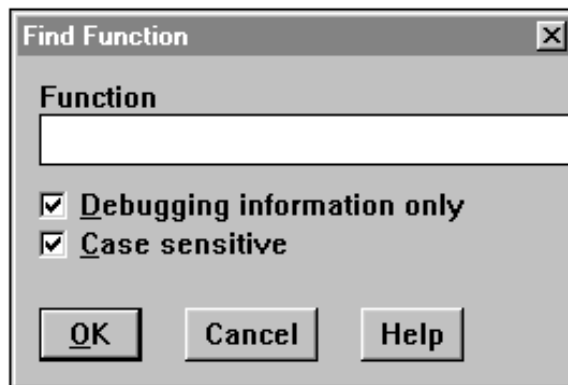


Figure 56. Find Function Window

To use the **Find Function** window:

- Type the name of the function you want to search for in the **Function** entry field.

If the function that you specify is not found, the following message displays:

No matching function found



This means it may be a static function or the function you specified does not exist.

The debugger searches each object file for global functions that match the function name specified. If an object file contains the global function that was specified, then it will also search for any static function with the same name.

**Note:** To search for a function in a specific executable file, disable this check box and type the name of the executable file in the **Executable** entry field and type the name of the source file in the **Source** entry field.

- Enable the **Debugging information only** check box if you want to search only the object files that contain debugging information.
- Enable the **Case sensitive** check box if you want to search for the string exactly as typed. Disable this check box if you want to search for both uppercase and lowercase characters.
- Select the **OK** push button.

### Where is execution point

Opens a source window containing the next line to be executed.

## Introducing the Main Debugging Windows

**Startup...** Displays the **Startup Information** window, which allows you to start another debugging session. Refer to “Debugging Win32 Applications on Windows NT or Windows 95” on page 265 for detailed information.

**Process list...** Displays the **Process List** window, which allows you to attach to processes.

**Note:** This choice is not available for Windows 95.

**Close debugger** Ends the debugging session. When you select the **Close debugger** choice, the **Close Debugger** message box prompts you to confirm that you want to end the debugging session.

### Breakpoints Menu Choices

Select choices from the **Breakpoints** menu to set breakpoints and to stop the execution of your program at any point. You can set as many breakpoints as you want.

Breakpoints can be set from the **Debug Session Control** window or from the source windows. When you set a breakpoint in one source window, it is reflected in the other source windows.

**Set line...** Displays the **Line Breakpoint** window, which allows you to set a line breakpoint to stop the execution of your program at a specific line number.

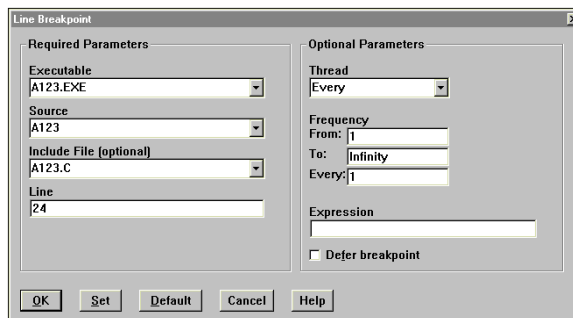


Figure 57. Line Breakpoint Window

The **Line Breakpoint** window is divided into two group headings: **Required parameters** and **Optional parameters**.

## Introducing the Main Debugging Windows

The **Required parameters** group heading contains the following:

- **Executable Entry Field**

To select a component from the **Executable** list:

1. Type the executable name in the entry field *or* open the **Executable** list by selecting the list button. This displays a drop-down combination box.
2. Highlight the executable where you want to set the breakpoint

- **Source Entry Field**

To select a component from the **Source** list:

1. Type the source name in the entry field *or* open the **Source** list by selecting the list button. This displays a drop-down combination box.
2. Highlight the source where you want to set the breakpoint.

- **Include File Entry Field**

If the source you selected has include files with executable statements, then the **File** list displays all the file names that contain executable lines.

1. Type the name of the file in the entry field *or* open the **File** list by selecting the list button. This displays a drop-down combination box.
2. Highlight the file where you want to set the breakpoint.

- **Line number Entry Field**

To set a line breakpoint, type the line number in the **Line number** entry field. The breakpoint is set on the line number.

The **Optional parameters** group heading contains the following:

- **Thread Entry Field**

To select a thread ID from the **Thread** list:

1. Open the **Thread** list by selecting the list button. This displays a drop-down combination box.
2. Highlight the thread where you want to set the breakpoint.

Select **EVERY**, the default, to set a breakpoint in all of the active threads in your program. The **Every** choice is thread independent. Select one of the individual threads to set a breakpoint in one thread only. Threads are added to the **Thread** list as new threads are activated.

- **From Entry Field**

This field is used for location breakpoints, address breakpoints and load occurrence breakpoints. Type in a whole integer number to start activating the breakpoint the *nth* time the location is encountered.

- **To Entry Field**

This field is used for location breakpoints, address breakpoints and load occurrence breakpoints. Type in a whole integer number to stop activating the breakpoint after the *nth* time the location is encountered.

## Introducing the Main Debugging Windows

- **Every Entry Field**

This field is used for location breakpoints, address breakpoints and load occurrence breakpoints. Type in a whole integer number to indicate how often the breakpoint should be activated within the **From** and **To** range.

- **Expression Entry Field**

If you are setting an address, function, or line breakpoint, you can also type in an expression. The execution of the program stops only if this condition tests true. For example, you could type the following:

```
(i==1) || (j==k) && (k!=5)
```



**Note:** Variables in a conditional expression associated with a FUNCTION breakpoint are limited to any static or global variables that are known to the called function when the function is called. Local variables and automatic variables cannot be used.

The maximum length of the condition is 256 characters.

- **Defer breakpoint Check Box**

Enable the **Defer breakpoint** check box if you want to set a breakpoint in a DLL that is not currently loaded.

**Note:** If your application consists of an EXE or preloaded DLLs, do not use this choice. If your application consists of DLLs that are dynamically loaded, use this choice to set breakpoints in DLLs which have not been loaded yet.

If you set a deferred line breakpoint and the line is located in a template, the debugger sets the line breakpoint in all of the templates when the DLL is loaded.

When a DLL is loaded and a deferred breakpoint has been set in the DLL, the state of the breakpoint changes from deferred to active. When a DLL is freed, any breakpoints that were set in the DLL change from the active state to deferred state.

If you enter an invalid source, file or line number, the debugger will be unable to activate the breakpoint when the DLL is loaded. Therefore, the invalid breakpoint will remain in the deferred state even after the DLL is loaded.

**Set function...** Displays the **Function Breakpoint** window, which allows you to set a function breakpoint to stop the execution of your program when the first instruction of the function is encountered where the breakpoint has been set.

## Introducing the Main Debugging Windows

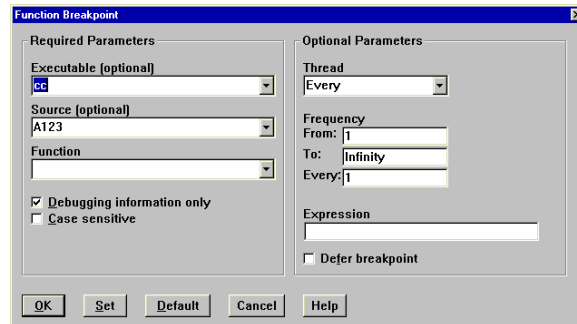


Figure 58. Function Breakpoint Window

The entry fields in this window are the same as in the **Set Line** window except for the following:

- **Function Entry Field**

Type the name of the function where you want to set the breakpoint or select a function from the **Function** list. To select a function, do the following:

1. Open the **Function** list by selecting the list button. This displays a drop-down combination box.
2. Highlight the function where you want to set the breakpoint.

If a function is overloaded, then a window displays with a list of all the overloaded function names. Select the appropriate function from the list.

- **Defer breakpoint Check Box**

Enable the **Defer breakpoint** check box if you want to set a breakpoint in DLLs which have not been loaded yet.

If you set a deferred breakpoint in a function and that function is overloaded, the debugger sets the breakpoint in all of the overloaded functions when the DLL is loaded.

When a DLL is loaded and a deferred breakpoint has been set in the DLL, the state of the breakpoint changes from deferred to active. When a DLL is freed, any breakpoints that were set in the DLL change from the active state to deferred state.

If you enter an invalid source file or invalid function, the debugger will be unable to activate the breakpoint when the DLL is loaded. Therefore, the invalid breakpoint will remain in the deferred state even after the DLL is loaded.

For a description of the types of data you can enter in the entry fields under the **Optional parameters** group heading, refer to “Set line...” on page 279.



## Introducing the Main Debugging Windows

**Set address...** Displays the **Address Breakpoint** window, which allows you to set an address breakpoint to stop the execution of your program at a specific address.

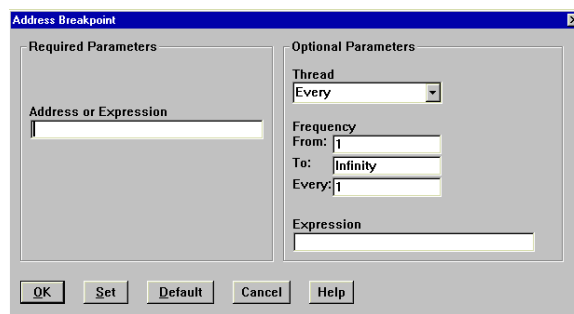


Figure 59. Address Breakpoint Window

The entry fields in this window are the same as in the **Set Line** window except for the following:

- **Address or Expression Entry Field**

Type the name of the address or expression in the **Address or Expression** entry field.

For example, to set an address breakpoint for the address `0x000A1FCC`, you would type one of the following in the **Address** entry field.

`0x000A1FCC` or `A1FCC`



The `0x` is optional.

For a description of the types of data you can enter in the entry fields under the **Optional parameters** group heading, refer to “Set line...” on page 279.

**Set change address...** Displays the **Change Address Breakpoint** window, which allows you to set a change address breakpoint to stop the execution of your program when contents of memory at a given address changes.

## Introducing the Main Debugging Windows

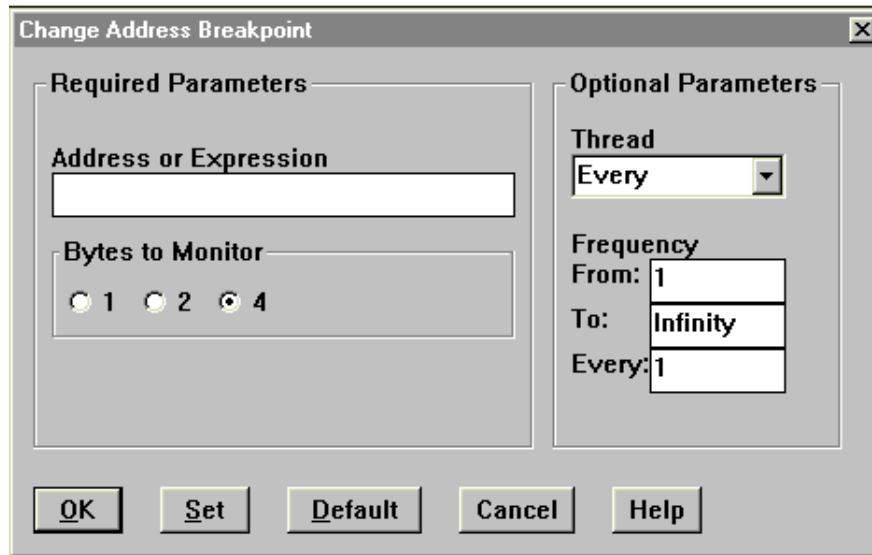


Figure 60. Change Address Breakpoint Window

Use the **Change Address Breakpoint** window to set a change address breakpoint.

- **Address or expression Entry Field**

Type a hexadecimal address or an expression that can be evaluated to a hexadecimal address.

**Note:** If you type *ABC* in the **Address or expression** entry field, and there is a variable named *ABC*, it uses the value of the variable instead of the hex value *ABC*. Also, you can type *&a* in the **Address or expression** entry field to set the breakpoint on the address of a variable *a*.

For example, type the following in the **Address or expression** entry field to set a change address breakpoint for the address *A1FCC*.



A1FCC

Type the following in the **Address or expression** entry field to set a change address breakpoint for the expression *&variable*.



&variable

## Introducing the Main Debugging Windows

**Warning:** If you set a change address breakpoint that is on the call stack, you should remove the breakpoint before leaving the routine associated with the breakpoint. Otherwise, when you return from the routine, the routine's stack frame will be removed from the stack leaving the breakpoint intact. Any other routine that gets loaded on the stack will then contain the breakpoint. You can set up to four breakpoints

For a description of the types of data you can enter in the entry fields under the **Optional parameters** group heading, refer to "Set line..." on page 279.

### Set load occurrence...

Displays the **Load Occurrence Breakpoint** window, which allows you to set a load occurrence breakpoint to stop the execution of your program when the DLL is encountered where the breakpoint has been set.

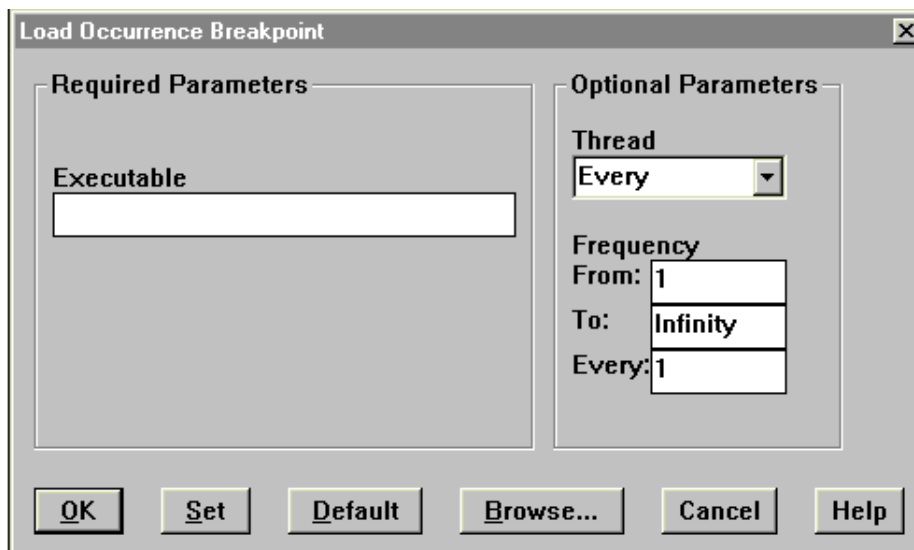


Figure 61. Load Occurrence Breakpoint Window

To use the **Load Occurrence Breakpoint** window, type the name of the DLL in the **File** entry field. Execution stops when the DLL is loaded.

- **File Entry Field**

To set a load occurrence breakpoint when MY.DLL is loaded, you would type one of the following in the **File** entry field:

MY or MY.DLL



## Introducing the Main Debugging Windows

For a description of the types of data you can enter in the entry fields under the **Optional parameters** group heading, refer to “Set line...” on page 279.

### List

Displays the **Breakpoint List** window, which lists all the breakpoints that have been set. It also shows the state of each breakpoints.

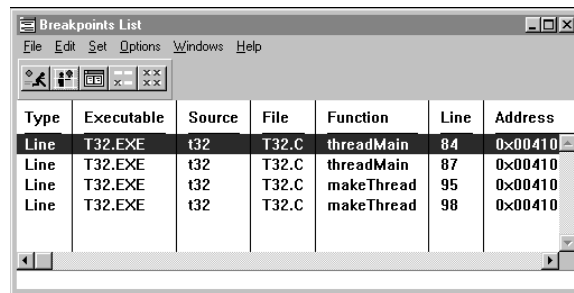


Figure 62. Breakpoint List Window

Use the **Breakpoints List** window to display a list of the breakpoints that have been set. The following information is also provided for each breakpoint.

- The enablement state
- The type of breakpoint
- The position of the breakpoint
- The conditions under which the breakpoint is activated.

For more information on the **Breakpoint List** window, refer to “Using the Breakpoint List Window” on page 316.

### Delete All

Deletes all the breakpoints that have been set.

## Monitors Menu Choices

Select choices from the **Monitors** menu of the **Debug Session Control** window to open other debugging windows such as monitors, call stack, registers, and storage.

The first three choices listed under the **Monitors** menu are also accessible from the tool bar of the source windows.

### Call stack

Displays the **Call Stack** window, which allows you to monitor the call stack for a particular thread. This window is described in “Using the Call Stack Window” on page 305.

## Introducing the Main Debugging Windows

<b>Registers</b>	Displays the <b>Registers</b> window, which allows you to monitor registers and flags for a particular component or thread. This window is described in “Using the Register Window” on page 308.
<b>Storage</b>	Displays the <b>Storage</b> window, which allows you to monitor the storage in your program. This window is described in “Using the Storage Window” on page 309.
<b>Local variables</b>	Displays the <b>Local Variables</b> window, which allows you to display the local variables for the program’s current function. This window is described in “Using the Local Variable Window” on page 313.

### Run Menu Choices

Select choices from the **Run** menu to execute your program, stop execution, or enable or disable threads.

**Run** Executes the program from the current line until a breakpoint is encountered or the program ends.

**Halt** Interrupts the program you are debugging.

**Hide Debugger on Run** Minimizes the debugger windows while your application is running.

### Check heap when stopping

Checks all memory blocks allocated or freed by the compiler debug memory management functions to make sure that overwriting has not occurred outside the bounds of allocated blocks and free memory blocks have not been overwritten. When **Check heap when stopping** choice is enabled, each time the program stops, the heap is checked. For example, stopping at a breakpoint or at the end of a step command would cause the heap check to be performed. If a heap error is detected, your application terminates. The **Termination** window displays showing the source line number where the application stopped and the heap check was performed.

### Notes

- For the **Check heap when stopping** choice to work, you have to compile your application using the *Tm+* compiler option.

## Introducing the Main Debugging Windows

- If you enable the **Check heap when stopping** choice and you run your application to termination, the heap check is not made. To check the heap just before termination, set a breakpoint on the last line of your application.
- If you are debugging a multiple thread program and a thread stops while running in compiler memory management code which is holding a memory semaphore, the heap check will not be performed.
- If the stopping thread is running in 16-bit code, the heap check will not be performed.

### Thread enabled

Enables or disables threads.

When a thread is enabled, a check mark symbol displays beside the **Thread enabled** choice and the thread is executed when you run your program. When a thread is not enabled, a check mark symbol does not display and the highlighted thread is not executed when you run your program.

## Options Menu Choices

Select choices from the **Options** menu to control how the debugger windows display.

### Window settings->

Use the **Window settings** cascading choices to modify the characteristics of the **Debug Session Control** window.

#### Fonts

Displays the **Fonts** window, which allows you to select the font you want to use for the text displayed in the **Debug Session Control** window.

#### Display style...

Displays the **Display Style** window which allows you to select the settings you want to use.

#### Restore defaults

Resets all of the window settings to their original settings.

#### Tool bar

Enable or disable the tool bar from being shown in the window.

#### Hover help

Enable or disable the hover help from being shown in the window.

#### Infoarea

Enable or disable the information area from being shown in the window.

### Debugger settings->

Use the **Debugger settings** cascading choice to set various debugger options that control how the debugger windows display. These settings affect the behavior of the debugger and remain in effect for the duration of the debugging session.

## Introducing the Main Debugging Windows

### Debugger properties...

Displays the **Debugger Properties** window, which allows you to select how the threads and source files initially display.

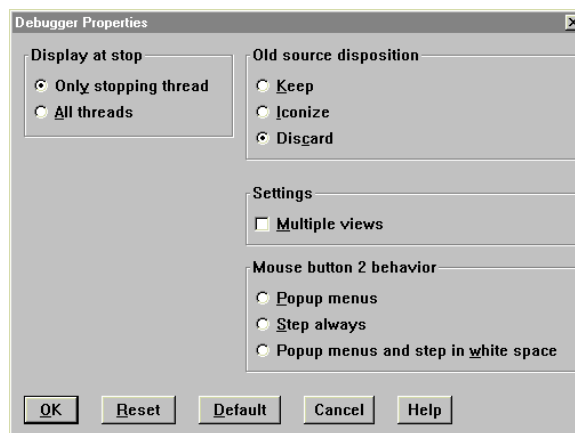


Figure 63. Debugger Properties Window

Use the **Debugger Properties** window to define the following:

- When a source window first displays during a debugging session.
- How to process a source window from which execution has just left. The window can remain displayed, be turned into an icon, or be discarded.

### Display at stop Group Heading

You can control how many source windows are displayed using the following radio buttons:

The choices are:

- |                             |                                                          |
|-----------------------------|----------------------------------------------------------|
| <b>Only stopping thread</b> | Show only the thread that caused to application to stop. |
| <b>All threads</b>          | Show where all threads are when the application stops.   |

For example, if you select **Only stopping thread**, the **Old source disposition** applies to all of the source windows except the current view of the stopping thread. If you select **All threads**, the **Old source disposition** applies only to the source windows for the components from which execution has just left within a thread.

### Old source disposition Group Heading

## Introducing the Main Debugging Windows

In the course of debugging, these selections allow you to control the behavior of source windows from which execution has just left. The **Old source disposition radio buttons** control the behavior of source windows within a thread.

The dispositions that the views can take are:

- |                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <b>Keep</b>    | Leaves open the source windows that contain the components and threads that you select with <b>Display at stop</b> . |
| <b>Iconize</b> | Changes into icons the views that contain the components and threads that you select with <b>Display at stop</b> .   |
| <b>Discard</b> | Disposes of the views that contain the components and threads that you select with <b>Display at stop</b> .          |

### Settings Group Heading

You can choose to display more than one source window for a particular source file. Enable the **Multiple views** check box if you want to have multiple source windows open at the same time.

### Mouse button 2 behavior Group Heading

Select the radio button that represents the action that you want mouse button 2 to perform.

### Monitor properties...

Displays the **Monitor Properties** window, which allows you to select the settings for monitoring variables or expressions.

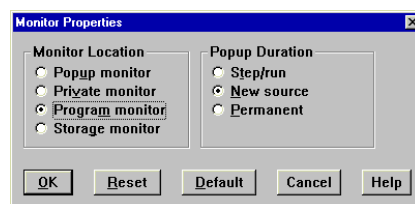


Figure 64. Monitor Properties Window

Use the **Monitor Properties** window to set the following:

- The window into which the variable or expression being monitored is placed.
- For popup expression windows, how long the monitor windows display.

### Monitor location Group Heading



## Introducing the Main Debugging Windows

Choose one of the following radio buttons to select the monitor window that opens when you select a variable or expression to monitor. The selections you can make, and the corresponding windows, are:

### **Popup**

Display the variable or expression in a popup expression window.

### **Private monitor**

Display the variable or expression in the **Private Monitor** window.

### **Program monitor**

Display the variable or expression in the **Program Monitor** window.

### **Storage monitor**

Display the variable or expression in the **Storage** window.

### **Popup duration Group Heading**

If you select **Popup** from the **Monitor location** group heading, select one of the following radio buttons to specify how long the popup expression window displays:

**Step/run**            The monitor window closes when the next step command or **Run** is executed.

**New source**        The monitor window closes when execution stops in a new source.

**Permanent**        This monitor window is associated with a specific source window and closes when the associated source window closes.

### **Default data representation->**

Use the **Default data representation** cascading choices to change the representation for a data type in a specific language.

Select a language to change the default representation of the selected data type. For example, you can change the default representation for an integer in the C language from decimal to hexadecimal. Select the **System** choice to change the default representation of the math coprocessor registers. This choice is language independent.

### **Program profiles->**

Use the **Program profiles** cascading choices to enable program profiles, delete program profiles, or change the location where the program profiles are stored,

*Program profiles* are used to restore the debugger windows and breakpoints when debugging a program more than once. They are stored separately for each program debugged. The file extension for the files that contain this information is @5R.

**Note:** Only information for executable files and preloaded DLLs relating to the primary thread is restored.

## Introducing the Main Debugging Windows

### Select information

Select what information you want saved in the program profiles.

### Delete program profiles

Delete program profiles for a program that you have debugged.

### Change location

Change the location of the file that holds the program profiles. This also moves any existing profiles to the new directory.

### Exception filtering...

Displays the **Exception Filtering** window, which allows you to select which exceptions you want the debugger to recognize.

To highlight an exception, do the following:

1. Select the exception by clicking on the name. It becomes highlighted.
2. Select the **OK** push button.

If a highlighted exception is encountered during the execution of your program, the **Application Exception** window is displayed. Any other exceptions that are encountered are ignored.

### Using the Application Exception Window

During execution of the program by the debugger, it is possible that the program can generate an exception. If this happens, the debugger suspends execution of the program and indicates the location within the code where the exception occurred.

The **Application Exception** window displays with the following choice:

<b>Examine/Retry</b>	You can investigate the cause of the exception and retry execution of the line that caused the fault.
<b>Step Exception</b>	The debugger steps into the first registered exception handler. Execution stops at the first executable line of code in that exception handler.
<b>Run Exception</b>	The debugger runs the exception handlers.

### Save window position and size

When you select this choice, the window positions and sizes are saved for each type of debugger window currently open.

### Global font change

Select the **Global font change** choice to change the font in all the debugger window.

When you select **Global font change**, the **Font Selection** window displays.

## Introducing the Main Debugging Windows

### Enable window cascading

Select the **Enable window cascading** choice to have the debugging windows overlap each other.

When you enable this choice, successive windows that are opened overlap each other on the screen. However, they are cascaded so that you can see the underlying windows. When this choice is disabled, successive windows cover previously displayed windows so that you do not see the underlying windows.

### Display tool buttons

Select the **Display tool buttons** choice if you want the tool buttons to be shown in **all** the debugger windows.

### Display hover help

Select the **Display hover help** choice if you want the hover help to be shown in **all** the debugger windows.

### Display Infoarea

Select the **Display infoarea** choice if you want the information area to be shown in **all** the debugger windows.

## Windows Menu Choices

Select the **Windows** menu of the **Debug Session Control** window to view a list of all the open debugger windows. By selecting a window from the **Windows** menu, it is brought into focus and made the active window. Also, if the window is minimized, it is restored.

## Help Menu Choices

Select choices from the **Help** menu of the **Debug Session Control** window to display the various types of help information.

<b>Help index</b>	Displays an alphabetical index of all available debugger help topics.
<b>General help</b>	Displays help information for the active window.
<b>Using help</b>	Describes how to use the help facility.
<b>How do I</b>	Displays the debugger task help.
<b>Product information</b>	Displays product information.

---

## Using the Source Windows

A source window allows you to view the program you are debugging. You can look at your source in one of the following windows:

- **Source**
- **Disassembly**
- **Mixed.**

## Introducing the Main Debugging Windows

A source window is thread-specific. Executable lines initially display in blue and non-executable lines initially display in black. Lines with breakpoints have a red prefix and lines with disabled breakpoints have a green prefix.

The **Source** window displays the source code for the object that contains the main function to the program being debugged. If it is available, the **Source** window displays with the **Debug Session Control** window when the debugging session starts. Otherwise, the **Disassembly** window displays.

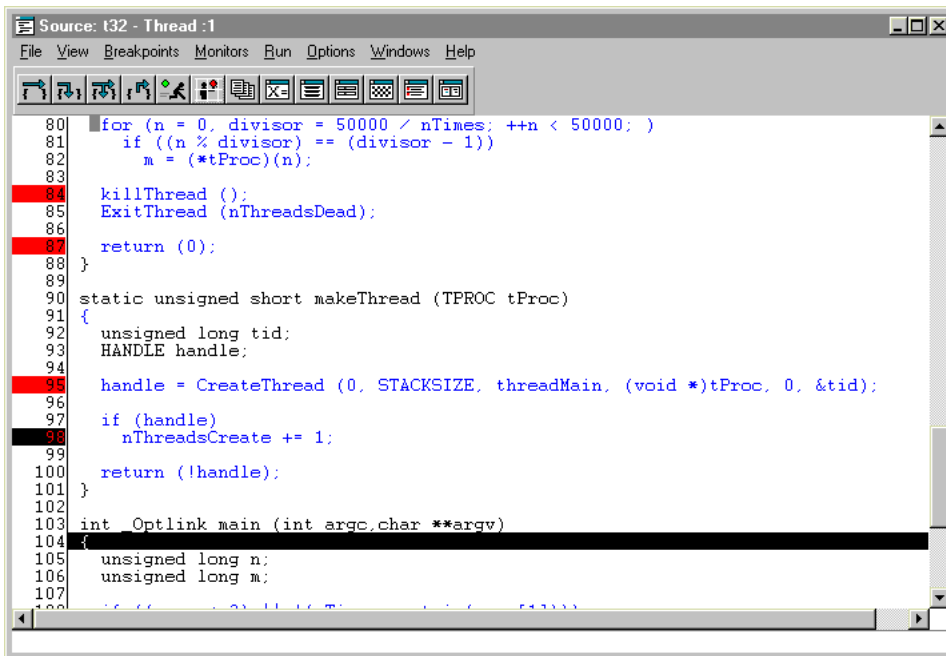


Figure 65. Source Window

The **Disassembly** window displays the assembler instructions for your program, without symbolic information.

## Introducing the Main Debugging Windows

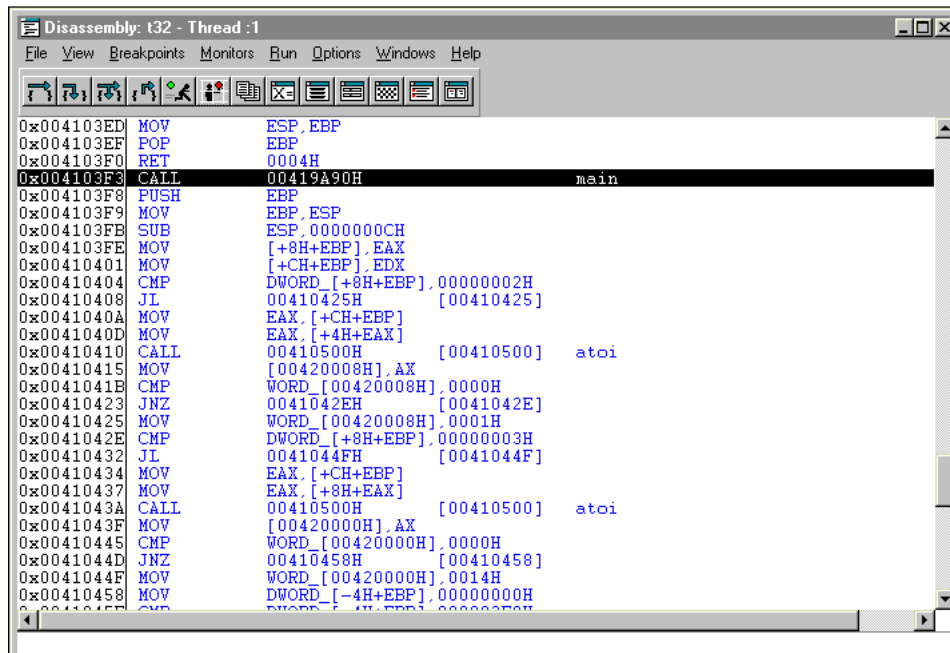


Figure 66. Disassembly Window

The **Mixed** window displays your program as follows:

- Each line of source code is prefixed by its line number, as in the **Source** window.
- Each disassembled line is prefixed by an address, as in the **Disassembly** window.
- Source comment lines also display.
- The lines of source code are treated as comments within the lines of disassembly code. You can only set breakpoints or run your program on lines of disassembly code.

**Note:** The **Mixed** window cannot be opened if the source code is not available.

## Introducing the Main Debugging Windows

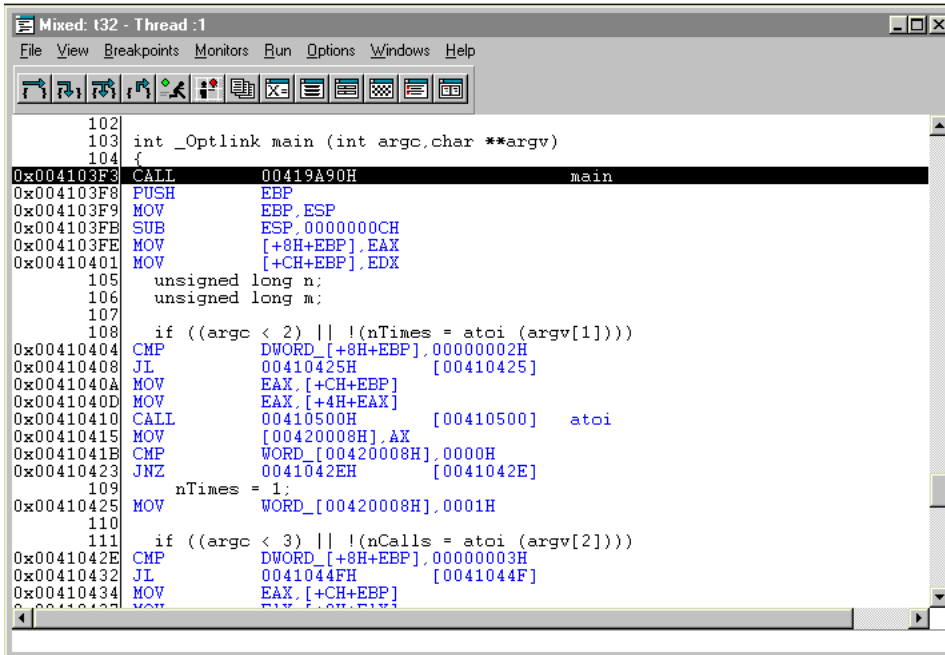


Figure 67. Mixed Window

### File Menu Choices

The choices in the **File** menu are the same as those listed under the **File** menu of the **Debug Session Control** window. Refer to “File Menu Choices” on page 277 for a description of the choices.

### View Menu Choices

Select choices from the **View** menu to locate strings of text, scroll to a particular line, view include files, change the text file or select a different view of your program.

**Find Window** Displays the **Find** window which allows you to search for a text string.

## Introducing the Main Debugging Windows



Figure 68. Find Window

To use the **Find** window to search for a text string:

1. Type the text string you want to search for in the **Enter text** entry field.
2. Enable the **Case sensitive** check box if you want to search for the string exactly as typed. Disable this check box to search for uppercase and lowercase characters.
3. Select the **OK** push button.

The search string can have alphabetic and numeric characters, a maximum of 256 characters, and uppercase and lowercase characters.

**Find next** Allows you to search for the next occurrence of the text string that you typed in the **Find** window.

**Scroll to line number...** Displays the **Scroll to Line Number** window, which allows you to go to a particular line in your program or set a line breakpoint.

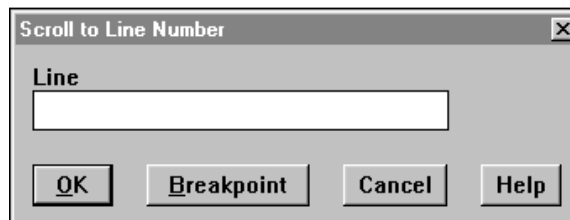


Figure 69. Scroll to Line Number Window

To use the **Scroll to Line Number** window to scroll to a specific line:

1. Type the line number you want to scroll to in the **Enter line number** entry field.
2. Select the **OK** push button to scroll to that line.

## Introducing the Main Debugging Windows

**Note:** If the **Source** window is active, just type a number and the **Scroll to Line Number** window automatically displays.

To use the **Scroll to Line Number** window to set a breakpoint:

1. Type the line number you want to set the breakpoint on in the **Enter line number** entry field.
2. Select the **Set Breakpoint** push button to set the breakpoint on the specified line number.

### Select include file

Displays the **Select Include File** window that allows you to view the files that are included in your program.

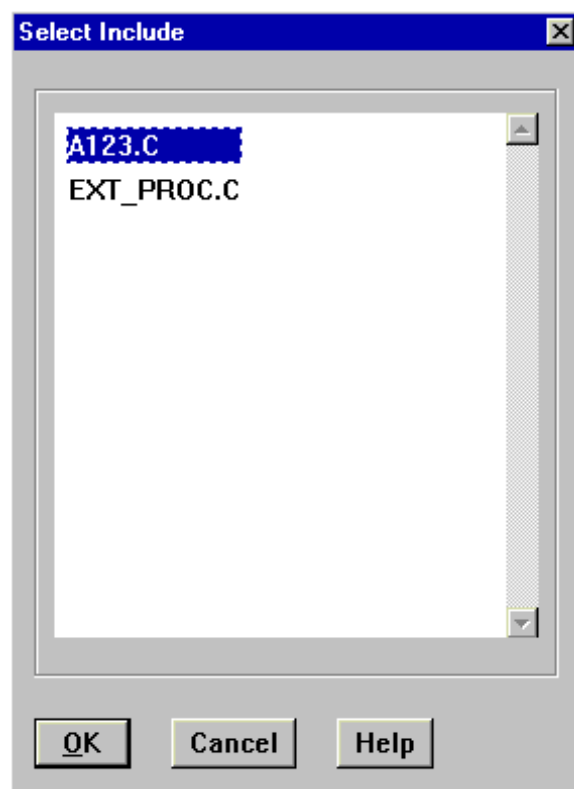


Figure 70. Select Include File Window

To use the **Select Include File** window:

1. Select the include file. The include file name is highlighted.
2. Select the **OK** push button. The selected include file view displays.



## Introducing the Main Debugging Windows

**Change text file** Displays the **Change Text File** window, which allows you to specify a file name to be used as the source in the current view. This is useful if the debugger found the incorrect source file for your program, so that you can specify the use of a different source file from a different directory.

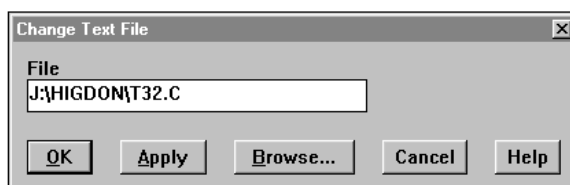


Figure 71. Change Text File Window

Use the **Change Text File** window to replace the path name or file name of the program you are debugging with a new path name or file name.

To replace the file name:

1. Type the new path name and file name in the **File name** entry field.
2. Select the appropriate push button.

**Source** Displays the **Source** window, which displays the source code for the object that contains the main function to the program being debugged.

Refer to “Using the Source Windows” on page 293 for more information.

**Disassembly** Displays the **Disassembly** window, which displays the assembler instructions for your program, without symbolic information.

Refer to “Using the Source Windows” on page 293 for more information.

**Mixed** Displays the **Mixed** window, which is a combination of the **Source** and **Disassembly** windows.

Refer to “Using the Source Windows” on page 293 for more information.

### Breakpoints Menu Choices

The choices listed under the **Breakpoints** menu are the same as those listed for the **Debug Session Control** window except for the **Toggle at current line** choice.

The **Toggle at current line** choice sets a breakpoint on the current line or deletes an existing breakpoint from the current line.

## Introducing the Main Debugging Windows

For a description of the other menu choices, refer to “Breakpoints Menu Choices” on page 279.

### Monitors Menu Choices

Select choices from the **Monitors** menu of the source windows to monitor expressions or variables and view the other debugger windows such as call stack, registers, and so on.

The first four choices listed under the **Monitors** menu are also accessible from the tool bar in the source windows.

### Monitor expression

Displays the **Monitor Expression** window, shown in Figure 72, which allows you to monitor expressions or variables and add them to various monitor windows.

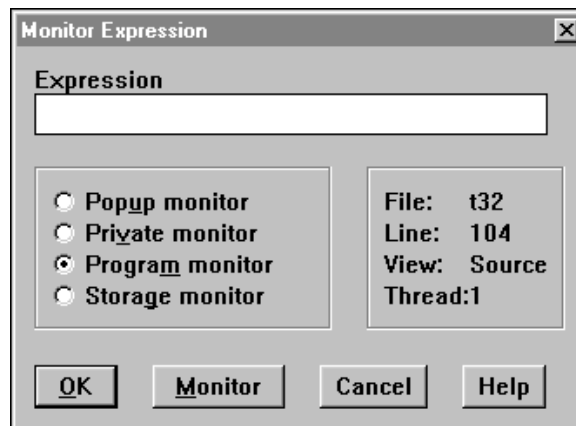


Figure 72. Monitor Expression Window

Use the **Monitor Expression** window to type the name of the expression you want to monitor. This window lists the following contextual information:

- The component you are in.
- The active line of the source code, which is highlighted.
- The view of the program that is active.
- The thread you are in.

To specify an expression to be monitored:

1. Type the name of the variable or expression you want to monitor in the **Expression** entry field.
2. Select the appropriate radio button from the **Add to** group heading for the location from where you want to monitor your expression.

## Introducing the Main Debugging Windows

**Note:** The expression displays as specified in the **Monitor Properties** window. To change the default location, select **Monitor properties** from the **Debugger settings** choice from the **Options** menu in the source windows or the **Debug Session Control** window.

<b>Call stack</b>	Displays the <b>Call Stack</b> window, which allows you to monitor the call stack for a particular thread. This window is described in “Using the Call Stack Window” on page 305.
<b>Registers</b>	Displays the <b>Registers</b> window, which allows you to monitor registers and flags for a particular component or thread. This window is described in “Using the Register Window” on page 308.
<b>Storage</b>	Displays the <b>Storage</b> window, which allows you to monitors the storage in your program. This window is described in “Using the Storage Window” on page 309.
<b>Local variables</b>	Displays the <b>Local Variables</b> window, which allows you to display the local variables for the program’s current function. This window is described in “Using the Local Variable Window” on page 313.

### Run Menu Choices

Select choices from the **Run** menu to perform step commands, run your program, hide debugger windows, and enable or disable threads.

<b>Step over</b>	Executes the current, highlighted line in the program, but does not enter any called function.
<b>Step into</b>	Executes the current, highlighted line in the program and enters any called program or function.
<b>Step debug</b>	Executes the current, highlighted line in the program. The debugger steps over any function for which debugging information is not available (for example, library and system routines), and steps into any function for which debugging information is available.
<b>Step return</b>	Automatically executes the lines of code up to, and including, the return statement of the current function.
<b>Run</b>	Runs the program, executing all enabled threads. Control returns to the debugger when the program ends or execution stops at an enabled breakpoint.
<b>Halt</b>	Interrupts the program you are debugging.

## Introducing the Main Debugging Windows

### Run to location

Executes your program from the current line up to the line that is highlighted or black in the prefix area.

To use the **Run to location** choice:

1. Single-click in the prefix area of the line you want to become the current line. The prefix area turns black.
2. Select the **Run to location** choice. The program runs up to the line that you marked.

The **Run to location** choice stops only on executable lines. If a highlighted line is not executable, the run is not performed.

### Jump to location

Select the **Jump to Location** choice to change the current line in your program without executing the lines between the present current line and the new current line.

To use the **Jump to location** choice:

1. Single-click in the prefix area of the line you want to become the current line. The prefix area turns gray.
2. Select the **Jump to location** choice. The current line is changed and the lines between are not executed.

The **Jump to location** choice stops only on executable lines. If a highlighted line is not executable, the jump is not performed.

**Warning:** Jumping out of the current function may corrupt the call stack and cause unpredictable results.

### Hide debugger on Run

For a description of this choice, refer to “Hide Debugger on Run” on page 287.

### Check heap when stopping

Checks all memory blocks allocated or freed by the compiler debug memory management functions to make sure that overwriting has not occurred outside the bounds of allocated blocks and free memory blocks have not been overwritten. When **Check heap when stopping** choice is enabled, each time the program stops, the heap is checked. For example, stopping at a breakpoint or at the end of a step command would cause the heap check to be performed. If a heap error is detected, your

## Introducing the Main Debugging Windows

application terminates. The **Termination** window displays showing the source line number where the application stopped and the heap check was performed.

### Notes

- For the **Check heap when stopping** choice to work, you have to compile your application using the *Tm+* compiler option.
- If you enable the **Check heap when stopping** choice and you run your application to termination, the heap check is not made. To check the heap just before termination, set a breakpoint on the last line of your application.
- If you are debugging a multiple thread program and a thread stops while running in compiler memory management code which is holding a memory semaphore, the heap check will not be performed.
- If the stopping thread is running in 16-bit code, the heap check will not be performed.

### Thread enabled

For a description of this choice, refer to “Thread enabled” on page 288.

### Options Menu Choices

Select choices from the **Options** menu to control how the debugger windows display.

### Window settings->

Use the **Window settings** cascading choices to modify the font of the source windows.

<b>Fonts...</b>	Displays the <b>Font Selection</b> window.
<b>Restore defaults</b>	Resets all of the window settings to their original settings.
<b>Notebook</b>	Displays the source windows in a notebook format.
<b>Tool bar</b>	Enables or disables the tool bar from being shown in the window.
<b>Hover help</b>	Enables or disables the hover help from being shown in the window.
<b>Infoarea</b>	Enables or disables the information area from being shown in the window.

### Debugger settings->

Use the **Debugger settings** cascaded choices to set various debugger options that control how the debugger windows display. These settings affect the behavior of the debugger and remain in effect for the duration of the debugging session.

Refer to “Debugger settings->” on page 288 for a description of the choices.

## Introducing the Main Debugging Windows

### Windows Menu Choices

Select the **Windows** menu of the source windows to view a list of all the open debugger windows. By selecting a window from the **Windows** menu, it is brought into focus and made the active window. Also, if the window is minimized, it is restored.

### Help Menu Choices

Select choices from the **Help** menu of the source windows to display the various types of help information.

Refer to “Help Menu Choices” on page 293 for a description of the help choices.

## Introducing the Other Debugging Windows

This section introduces the basic debugging windows. These are the windows that are normally located in the **Monitors** menu of the **Debug Session Control** and the source windows. The windows include call stack, registers, storage, monitor, and breakpoints.

### Using the Call Stack Window

The **Call Stack** window as shown in Figure 73, lists all of the active functions for a particular thread including the system calls. The functions are displayed in the order that they were called.

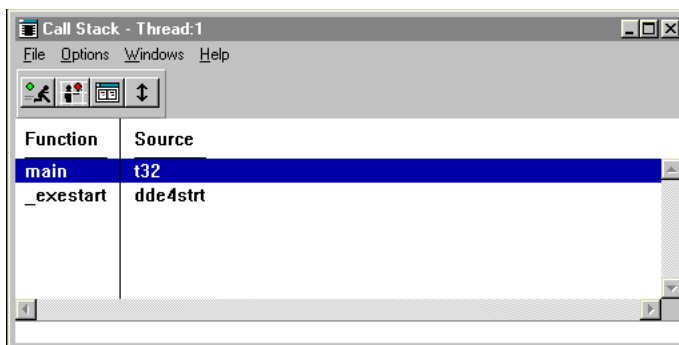



Figure 73. Call Stack Window

Each **Call Stack** window displays call stack information for one thread only. When the state of the program changes, such as when you execute the program or you update displayed data, the **Call Stack** window changes to reflect the current state. You can double-click on any call stack entry to display the source code for that entry. The line that calls the next stack entry is highlighted. The remaining stack size shows the bytes left in the stack for the thread.

**Note:** The stack may not display correctly if you do not follow the standard calling conventions or if you step into optimized code.

To display the **Call Stack** window, select **Call Stack** from the **Monitors** menu or select the **Call Stack** button  from the tool bar.

## Introducing the Other Debugging Windows

### File Menu Choice

Use the choice from the **File** menu to end the debugging session.

The **Close debugger** choice allows you to end the current debugging session. When you select **Close debugger**, the **Close Debugger** message box prompts you to confirm that you want to end the debugging session.

### Options Menu Choices

Use choices from the **Options** menu to select the font you want for the **Call Stack** window and control how the items on the call stack display.

#### Fonts...

Displays the **Fonts Selection** window, which allows you to select the type of font you want to use for the **Call Stack** window.

#### Display style...

Displays the **Display Style** window, shown in Figure 74, which allows you to select the type of information you want displayed in the call stack and choose how the items are to be displayed.

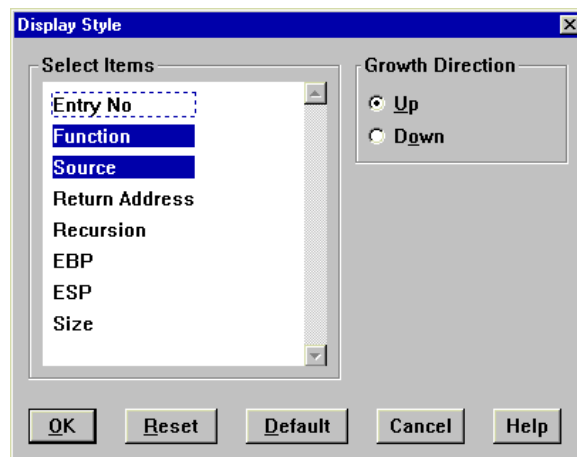


Figure 74. Display Style Window

To use the **Display Style** window:

1. Select one or more of the items under the **Select items** to display for each call stack entry. Each item causes a new column to be added to the **Call Stack** window.

The following items are available:



## Introducing the Other Debugging Windows

<b>Entry No.</b>	Represents the position of the call stack item in the list. Entry level 1 is the first function invoked.
<b>Function</b>	Lists program name or the address of the function call that created the new call stack entry.
<b>Source</b>	Lists the component name that contains the function. The name displayed corresponds with a name listed in the <b>Components</b> list box in the <b>Debug Session Control</b> window.
<b>Return Address</b>	The address represents where execution will return in that function.
<b>Recursion</b>	Lists the recursion level. 0 is the first invocation.
<b>Base Pointer (EBP)</b>	Start of the call stack frame for that function.
<b>Stack Pointer (ESP)</b>	End of the call stack frame for that function.
<b>Size</b>	Size of the call stack frame for that function.

2. Select one of the following **Growth direction** radio buttons to determine how new items are displayed on the call stack.

<b>Up</b>	Displays new items at the top of the <b>Call Stack</b> window.
<b>Down</b>	Displays new items at the bottom of the <b>Call Stack</b> window.

**Restore defaults** Select the **Restore defaults** choice to reset all of the window settings to their original settings.

**Tool bar** Select the **Tool bar** choice to enable or disable the tool bar from being shown in the window.

If enabled, the tool bar is displayed in the window. If disabled, the tool bar is not displayed.

### Windows Menu Choices

Select the **Windows** menu of the **Call Stack** window to view a list of all the open debugger windows. By selecting a window from the **Windows** menu, it is brought into focus and made the active window. Also, if the window is minimized, it is restored.

### Help Menu Choices

Select choices from the **Help** menu of the **Call Stack** window to display the various types of help information.

## Introducing the Other Debugging Windows

Refer to “Help Menu Choices” on page 293 for a description of the help choices.

---

### Using the Register Window

The **Register** window, as shown in Figure 75, lists all the processor registers for a particular thread. The contents of all of the registers except floating-point registers are displayed in hexadecimal. To update a register, double-click on the register and a multiple-line entry field displays. Type over the contents and press Enter.

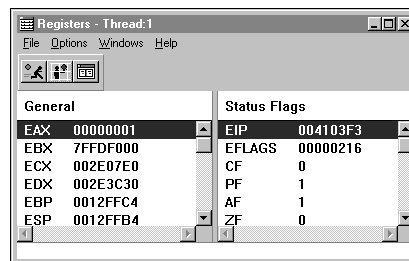



Figure 75. Registers Window

In the **Registers** window, floating-point registers display as floating-point decimal numbers. They can be updated with a floating-point decimal number or with a hexadecimal string that represents a floating-point number.

To display the processor registers and flags, select **Registers** from the **Monitors** menu or select the **Register** button  from the tool bar.

### File Menu Choices

Use the choice from the **File** menu to end the debugging session.

The **Close debugger** choice allows you to end the current debugging session. When you select **Close debugger**, the **Close Debugger** message box prompts you to confirm that you want to end the debugging session.

### Options Menu Choice

Use the choices from the **Options** menu to select the font you want for the **Registers** window, select the items you want shown in the window, restore the defaults, and enable or disable the tool bar.

### Fonts...

When you select **Fonts...**, the **Font Selection** window displays. From this window, select the font you want shown in the **Registers** window.

## Introducing the Other Debugging Windows

<b>Display style...</b>	Displays the <b>Display Style</b> window which allows you to select the type of information you want displayed in the <b>Registers</b> window.
<b>Restore defaults</b>	Select the <b>Restore defaults</b> choice to reset all of the window settings to their original settings.
<b>Tool bar</b>	<p>Select the <b>Tool bar</b> choice to enable or disable the tool bar from being shown in the window.</p> <p>If enabled, the tool bar is displayed in the window. If disabled, the tool bar is not displayed.</p>
<b>Hover help</b>	<p>Select the <b>Hover help</b> choice to enable or disable hover help from being shown in the window.</p> <p>If enabled, the hover help is displayed in the window. If disabled, the help is not displayed.</p>
<b>Infoarea</b>	<p>Select the <b>Infoarea</b> choice to enable or disable the information area from being shown in the window.</p> <p>If enabled, the infoarea is displayed in the window. If disabled, the infoarea is not displayed.</p>

### Windows Menu Choices

Select the **Windows** menu from the **Registers** window to view a list of all the open debugger windows. By selecting a window from the **Windows** menu, it is brought into focus and made the active window. Also, if the window is minimized, it is restored.

### Help Menu Choices

Select choices from the **Help** menu of the **Registers** window to display the various types of help information.

Refer to “Help Menu Choices” on page 293 for a description of the help choices.

---

## Using the Storage Window

The **Storage** window, as shown in Figure 76 on page 310, shows the storage contents and the address of the storage.

## Introducing the Other Debugging Windows

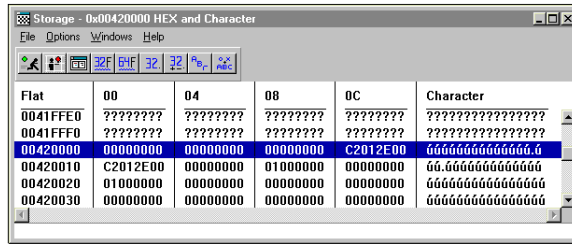


Figure 76. Storage Window

Multiple storage windows can display the same storage. When you run a program or update displayed data, the **Storage** window is updated to reflect the change.

To update the storage contents and all affected windows, double-click and in the multiple-line entry field that displays type over the contents of the field.

To specify a new address location, type over the address field in the **Storage** window. The window scrolls to the appropriate storage location.

To display the **Storage** window, select **Storage** from the **Monitors** menu or select the

**Storage** button  from the tool bar.

## File Menu Choice

Use the choice from the **File** menu to end the debugging session.

The **Close debugger** choice allows you to end the current debugging session. When you select **Close debugger**, the **Close Debugger** message box prompts you to confirm that you want to end the debugging session.

## Options Menu Choices

Use choices from the **Options** menu to monitor expressions, control how the items in the storage window display, and select the font you want for the **Storage** window.

**Fonts...** Displays the **Font Selection** window, which allows you to select the type of font you want to use for the **Storage** window.

<b>Display style...</b>	Displays the <b>Display Style</b> window, shown in Figure 77 on page 311, which allows you to select the format for the storage contents and storage addresses and change the columns per line that display.
-------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Introducing the Other Debugging Windows

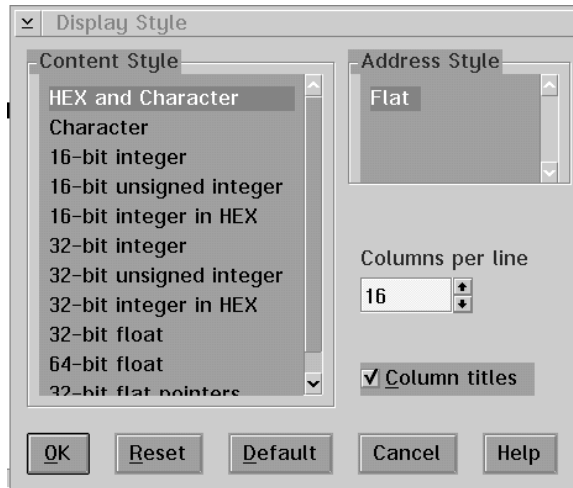


Figure 77. Display Style Window

Use the **Storage Display Style** window to select the parameters that control how the storage contents display and set how the storage addresses display.

### Content style Group Heading

Select how you want the storage contents displayed. You can select from several storage display styles.

To select the storage content style:

1. Scroll to the content style you want.
2. Select the content style.
3. The style becomes highlighted.

### Address style Group Heading

Select the available address style.

To select the address style:

1. Select the address style.
2. The address style becomes highlighted.

### Columns per line Entry Field

Select the number of columns per line you want displayed in the **Storage** window.

## Introducing the Other Debugging Windows

Use the Up or Down arrow keys to select the number of columns you want displayed in the **Storage** window. The available number of columns per line are 1-16.

Enable the **Column titles** check box if you want to display the titles of the columns in the **Storage** window.

### Monitor expression...

Displays the **Monitor Expression in Storage** window, as shown in Figure 78, which allows you to type in the name of the expression you want to monitor.

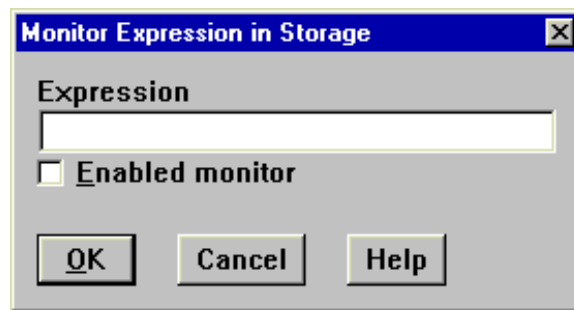


Figure 78. Monitor Expression in Storage Window

To specify an expression, type the name or address of the variable or expression you want to monitor in the **Address or expression** entry field.

The expression evaluator used is based on the context. For example if you display the **Storage** window by selecting the **Monitor expression...** choice from the **Monitors** menu, the evaluator used is based on the context in the **Monitor Expression** window. However, if you display the **Storage** window first and then select the **Monitor expression...** choice from the **Options** menu of the **Storage** window, the evaluator used is based on the context of the stopping thread.

**Note:** You cannot look at variables that have been defined using the `#DEFINE` preprocessor directive. If the variable is not in scope when the monitor is opened, the default address is displayed. If the variable goes out of scope, the address is changed to a hex constant.

If you enable the **Enabled monitor** check box, the monitor updates the stop value of the program to the actual value in storage. However, a disabled monitor suspends this updating and reflects the stop value or the value held when the monitor was disabled.

### Restore defaults

Select the **Restore defaults** choice to reset all of the window settings to their original settings.

## Introducing the Other Debugging Windows

**Tool buttons** Select the **Tool buttons** choice if you want the tool buttons to be shown in the window.

**Hover help** Select the **Hover help** choice if you want the hover help to be shown in the window.

**Infoarea** Select the **Infoarea** choice if you want the information area to be shown in the window.

### Windows Menu Choices

Select the **Windows** menu from the **Storage** window to view a list of all the open debugger windows. By selecting a window from the **Windows** menu, it is brought into focus and made the active window. Also, if the window is minimized, it is restored.

### Help Menu Choices

Select choices from the **Help** menu of the **Storage** window to display the various types of help information.

Refer to “Help Menu Choices” on page 293 for a description of the help choices.

---

## Using the Local Variable Window

The **Local Variables** window, as shown in Figure 79, monitors the local variables (static, automatic, and parameters) for the current execution point in the program. The contents of the **Local Variables** window change each time your program enters or leaves a function.

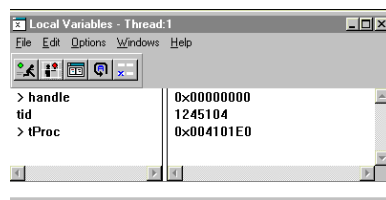


Figure 79. Local Variables Window

### File Menu Choice

Use the choice from the **File** menu to end the debugging session.

## Introducing the Other Debugging Windows

The **Close debugger** choice allows you to end the current debugging session. When you select **Close debugger**, the **Close Debugger** message box prompts you to confirm that you want to end the debugging session.

### Edit Menu Choices

Select the choices from the **Edit** menu of the **Local Variables** window to delete, select, deselect, show other elements or change representation of the variables.

#### Delete

Select the **Delete** choice to delete variables or expressions that are being monitored from a monitor window.

To delete a variable or expression from a monitor window:

1. Select the variable or expression using your mouse pointer. The variable or expression becomes highlighted.
2. Select the **Delete** choice from the **Options** menu.

#### Select all

Select the **Select all** choice to select all the expressions in the window.

#### Deselect all

Select the **Deselect all** choice to cancel the selection of all the expressions in the window,

#### Other elements...

Select the **Other elements** choice to view the next 50 items (classes, arrays, or structures) that are related to the variable or expression that you are monitoring.

### Representation->

Use the **Representation** cascading choices to display the contents of the variable in a new representation. The types of representation that display on the menu depend on the data type of the variable you are monitoring.

The following are possible representations:

#### Hexadecimal

Displays the contents of the monitored variable in hexadecimal notation.

#### Decimal

Displays the contents of the monitored variable in decimal notation.

#### String

Displays the contents of the monitored variable as a character string.

#### Hexadecimal pointer

Displays the contents of the monitored variable as a hexadecimal pointer.

#### Decimal pointer

Displays the contents of the monitored variable as a decimal pointer.



## Introducing the Other Debugging Windows

### Array

Displays the contents of the monitored variable as an array.

### Floating-point

Displays the contents of the monitored variable in floating-point notation.

### Character

Displays the contents of the monitored variable in character form.

**Note:** Floating-point registers or variables display as either a floating-point decimal number or as a hexadecimal string. However, they cannot be updated with a hexadecimal string that represents a floating-point number. If you need to update a floating-point variable with a hexadecimal representation of a floating-point number, you must step through the **Disassembly** window to see when the variable loads into a register and then change the value in the **Registers** window.

## Options Menu Choices

Select choices from the **Options** menu to control how the contents of variables display and set debugger options.

### Fonts...

Displays the **Font Selection** window, which allows you to select the type of font you want to use for the **Local Variables** window.

### Restore defaults

Select the **Restore defaults** choice to reset all of the window settings to their original settings.

### Show context

Select the **Show context** choice to display the contextual information for the variable you are monitoring. The following information displays:

- Source
- File
- Line
- Thread.

### Tool buttons

Select the **Tool buttons** choice if you want the tool buttons to be shown in the window.

### Hover help

Select the **Hover help** choice if you want the hover help to be shown in the window.

### Infoarea

Select the **Infoarea** choice if you want the information area to be shown in the window.

## Introducing the Other Debugging Windows

### Windows Menu Choices

Select the **Windows** menu from the **Local Variables** window to view a list of all the open debugger windows. By selecting a window from the **Windows** menu, it is brought into focus and made the active window. Also, if the window is minimized, it is restored.

### Help Menu Choices

Select choices from the **Help** menu of the **Local Variables** window to display the various types of help information.

Refer to “Help Menu Choices” on page 293 for a description of the help choices.

---

## Using the Monitor Windows

The debugger has three other windows that allow you to monitor variables and expressions. These windows are as follows:

- **Data Popup**
- **Program Monitor**
- **Private Monitor.**

A **Data Popup** window monitors single variables or expressions. This window is associated with a specific source window and closes when the associated window closes.

The variables or expressions can be transferred either to the **Program Monitor** window or the **Private Monitor** window.

The **Program Monitor** and the **Private Monitor** windows are used as collectors for individual variables or expressions you might be interested in. Variables and expressions may be created in these monitors or may be transferred to them from a **Data Popup** window.

The difference between the **Private Monitor** window and the **Program Monitor** window is the length of time that they remain open. The **Program Monitor** window remains open for the entire debugging session. The **Private Monitor** window is associated with the source window from which it was opened and closes when its associated view is closed.

---

## Using the Breakpoint List Window

Use the **Breakpoint List** window to display a list of the breakpoints that have been set. The following information is provided for each breakpoint that has been set:

- The type of breakpoint

## Introducing the Other Debugging Windows

- The position of the breakpoint
- The enablement state
- The conditions under which the breakpoint is activated.

To display the **Breakpoint List** window, as shown in Figure 80, select **List** from the **Breakpoints** menu or select the **Breakpoints** button in the tool bar.

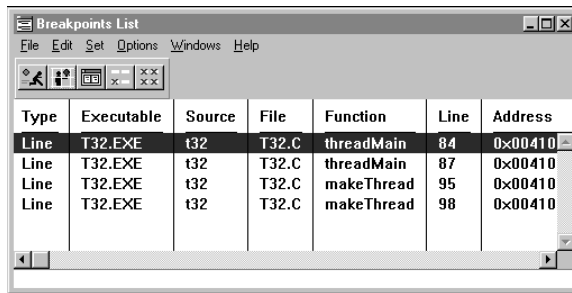


Figure 80. Breakpoint List Window

### File Menu Choice

Use the choice from the **File** menu to end the debugging session.

The **Close debugger** choice allows you to end the current debugging session. When you select **Close debugger**, the **Close Debugger** message box prompts you to confirm that you want to end the debugging session.

### Edit Menu Choices

Select the choices from the **Edit** menu of the **Breakpoint List** window to delete, disable, modify, or enable breakpoints.

#### Delete

Deletes any breakpoints that are highlighted in the **Breakpoint List** window.

To delete a breakpoint:

1. Highlight the breakpoint you want to delete.
2. Select the **Delete** choice.

#### Disable

Disables any highlighted breakpoints. The breakpoint remains set but not active. This allows you to run your program and not stop when the breakpoint is encountered.

To disable a breakpoint:

1. Highlight the breakpoint you want to disable.

## Introducing the Other Debugging Windows

2. Select the **Disable** choice.

### Modify

Use the **Modify** choice to change the breakpoints that have been set in your program.

To modify a breakpoint:

1. Highlight the breakpoint you want to change.
2. Select the **Modify** choice. The breakpoint window that represents the type of breakpoint displays.
3. Make the appropriate changes to the entry fields.
4. Select the **OK** push button to accept your changes and close the window. If you want to make other changes, select the **Set** push button to accept the changes and keep the window open.

### Delete all

Deletes all the breakpoints that have been set.

To delete all the breakpoints:

1. Select the **Delete all** choice. The **Delete All Breakpoints** window displays.
2. Select **Yes** from the **Delete All Breakpoints** window.

### Disable all

Disables all the breakpoints that have been set.

To disable all the breakpoints:

Select the **Disable all** choice. The prefix area in the source window changes from red to green to show that the breakpoints are disabled.

### Enable all

Enables all the breakpoints that have been disabled.

To enable the breakpoints:

Select the **Enable all** choice. The prefix area in the source window changes from green to red to show that the breakpoints are enabled.

## Set Menu Choices

Refer to “Breakpoints Menu Choices” on page 279 for a description of the **Set** menu choices.

## Options Menu Choices

Select choices from the **Options** menu of the **Breakpoint List** window to sort the breakpoints, change the style, and change the font for the window.

### Fonts...

Displays the **Font Selection** window that allows you to select the font you want to use for the text in the **Breakpoint List** window.

## Introducing the Other Debugging Windows

### Display style...

Displays the **Display Style** window, which allows you to control how the items in the breakpoint list display.

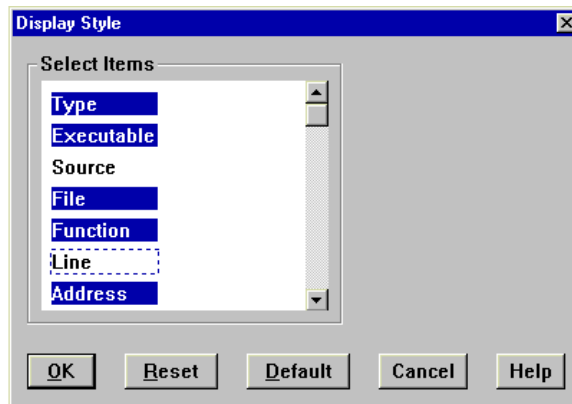


Figure 81. Breakpoints - Display Style Window

To change the columns which are displayed in the breakpoint list window:

1. Select one or more of the items under the **Columns** group heading. Each item you select causes a new column to be added to the **Breakpoint List** window.
2. Select the **OK** push button.

### Sort

Displays the **Sort Breakpoints** window, shown in Figure 82, which allows you to sort the breakpoints by the characteristics of the breakpoint.

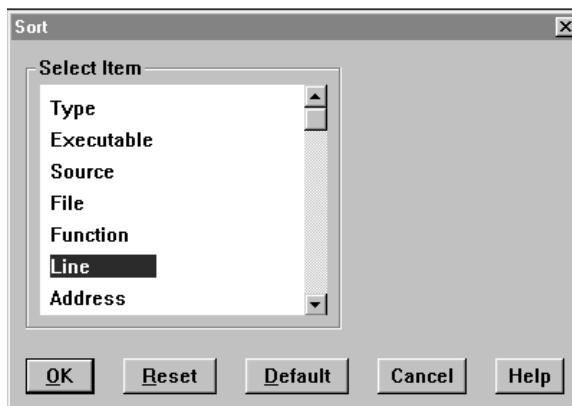


Figure 82. Sort Breakpoints Window

## Introducing the Other Debugging Windows

Use the **Sort Breakpoints** window to sort the breakpoints that have been set in your program.

Breakpoints can be sorted according to the following categories:

- Type
- Executable
- Source
- File
- Function
- Line
- Address
- State
- Status
- Thread
- Expression
- From
- To
- Every.

Select the category you want and select the **OK** push button.

### **Restore defaults**

Select the **Restore defaults** choice to reset all of the window settings to their original settings.

### **Tool buttons**

Select the **Tool buttons** choice if you want the tool buttons to be shown in the window.

### **Hover help**

Select the **Hover help** choice if you want the hover help to be shown in the window.

### **Infoarea**

Select the **Infoarea** choice if you want the information area to be shown in the window.

## **Windows Menu Choices**

Select the **Windows** menu from the **Breakpoint List** window to view a list of all the open debugger windows. By selecting a window from the **Windows** menu, it is brought into focus and made the active window. Also, if the window is minimized, it is restored.

## **Help Menu Choices**

Select choices from the **Help** menu of the **Breakpoint List** window to display the various types of help information.

## **Introducing the Other Debugging Windows**

Refer to “Help Menu Choices” on page 293 for a description of the help choices.

## **Introducing the Other Debugging Windows**





## Chapter 21. Expressions Supported

This section describes the expression language supported by the debugger, which is a subset of C/C++. This includes the operands, operators, and data types.

**Note:** You can display and update bit fields for C/C++ code only. You cannot look at variables that have been defined using the #DEFINE preprocessor directive.

---

### Supported Expression Operands

You can monitor an expression that uses the following types of operands only:

Operand	Definition
<b>Variable</b>	A variable used in your program.
<b>Constant</b>	The constant can be one of the following types: <ul style="list-style-type: none"><li>• Fixed or floating-point constant. <b>Note:</b> The largest floating-point constant is 1.8E308. The smallest floating-point is 2.23E-308.</li><li>• A string constant, enclosed in quotation marks ( " ")</li><li>• A character constant, enclosed in single quote marks ( ' ' )</li></ul>
<b>Registers</b>	In the case of conflicting names, the program variable names take precedence over the register names. For conversions that are done automatically when the registers display in mixed-mode expressions, general purpose registers are treated as unsigned arithmetic items with a length appropriate to the register.

If you monitor an enumerated variable, a comment displays to the right of the value. If the value of the variable matches one of the enumerated types, the comment contains the name of the first enumerated type that matches the value of the variable. If the length of the enumerated name does not fit in the monitor, the contents display as an empty entry field.

The comment (empty or not) lets you distinguish between a valid enumerated value and an invalid value. An invalid value does not have a comment to the right of the value.

You can *not* update an enumerated variable by entering an enumerated type. You must enter a value or expression. If the value is a valid enumerated value, the comment to the right of the value updates.

## Expressions Supported

Bit fields are supported for C/C++ compiled code only. You can display and update bit fields, but you cannot use them in expressions. You cannot look at variables that have been defined using the #DEFINE preprocessor directive.

---

## Supported Expression Operators

You can monitor an expression that uses the following operators only:

<i>Figure 83 (Page 1 of 2). Supported Expression Operators</i>	
Operator	Coded as
Global scope resolution	::a
Class scope resolution	a::b
Subscripting	a[b]
Member selection	a.b or a->b
Size	sizeof a or sizeof (type)
Logical not	!a
One's complement	~a
Unary minus	-a
Unary plus	+a
Dereference	*a
Type cast	(type) a
Multiply	a * b
Divide	a / b
Modulo	a % b
Add	a + b
Subtract	a - b
Left shift	a << b
Right shift	a >> b
Less than	a < b
Greater than	a > b
Less than or equal to	a <= b
Greater than or equal to	a >= b
Equal	a == b
Not equal	a != b
Bitwise AND	a & b
Bitwise OR	a   b
Bitwise exclusive OR	a ^ b

## Expressions Supported

<i>Figure 83 (Page 2 of 2). Supported Expression Operators</i>	
Operator	Coded as
Logical AND	$a \ \&\& \ b$
Logical OR	$a \    \ b$

---

## Supported Data Types

You can monitor an expression that uses the following typecasting operations:

- 8-bit signed byte
- 8-bit unsigned byte
- 16-bit signed integer
- 16-bit unsigned integer
- 32-bit signed integer
- 32-bit unsigned integer
- 32-bit floating-point
- 64-bit floating-point
- 128-bit floating-point
- Pointers
- User-defined types.

## Expressions Supported

---

## Part 6. Browsing Programs and Libraries

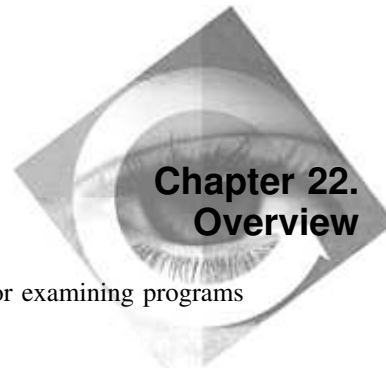
As a C++ programmer using the object-oriented paradigm, you must deal with large and sometimes complex collections of interrelated classes. A class' behavior is not only defined by its own data and function members, but also by all the behavior of its base classes. The VisualAge for C++ Browser is a tool to help you understand and use these classes and their relationships.

---

<b>Chapter 22. Overview</b> . . . . .	329
Understanding the Browser . . . . .	329
Concepts Used by the Browser . . . . .	330
Using the Mouse . . . . .	331
Getting Help While You Are Using the Browser . . . . .	331
 <b>Chapter 23. Getting Started</b> . . . . .	 333
Starting the Browser . . . . .	333
Creating Files to Use with the Browser . . . . .	335
Closing the Browser . . . . .	336
 <b>Chapter 24. Understanding and Using the Browser User Interface</b> . . . . .	 339
The List Window . . . . .	340
The Graph Window . . . . .	351
Changing Browser Settings . . . . .	363
Changing Fonts . . . . .	366
Loading Files into the Browser . . . . .	368
Merging Files . . . . .	369
Finding Objects in the Current Window . . . . .	371
Searching for Objects in the Entire Browser Database . . . . .	372
The History Window . . . . .	373
 <b>Chapter 25. Using the Browser</b> . . . . .	 375
Using the Browser to Assist in Development . . . . .	375
Using the Browser to Aid Program Understanding . . . . .	378
Using QuickBrowse . . . . .	389
Updating the Browser Database . . . . .	392
Adding Menu Items to the Load ► and Merge ► Cascade menus . . . . .	393
 <b>Chapter 26. A Tour of the Browser</b> . . . . .	 395
Starting the Browser and Loading User Interface Classes . . . . .	396
Finding A Class . . . . .	397
Showing the Inheritance Relationship of a Class . . . . .	398
Finding Another Class . . . . .	399

Changing the View of a Graph . . . . .	400
Investigating the Members of a Class . . . . .	401
Customizing Program Elements . . . . .	402
Editing Files from the Browser . . . . .	402
Organizing the Information in a List Window . . . . .	404
Finding A Function . . . . .	405
Showing the VisualAge for C++ Documentation for a Particular Function . . . .	406
More About the PopUp Menu Actions . . . . .	406
Invoking Actions Again . . . . .	407
Graphing Include File Relationships . . . . .	407
Returning to Previous Queries/Displays . . . . .	408
Keeping Your Windows From Being Replaced . . . . .	409
Changing the Default Settings for List and Graph Windows . . . . .	410
Manipulating Graphs . . . . .	410
The Browser and WorkFrame . . . . .	411
 <b>Chapter 27. Troubleshooting</b> . . . . .	 413
The Browser Won't Start . . . . .	413
Error Loading a .EXE, .DLL, or .LIB file . . . . .	413
Error Loading a .PDB File . . . . .	413
Adding Files to the Load ► and Merge ► Menus Doesn't Work . . . . .	413
The Graph Zone Will Not Maximum Zoom . . . . .	414
 <b>Chapter 28. Browser Fast-Path Keys and Menu Descriptions</b> . . . . .	 415
Fast-Path Keys . . . . .	415
PullDown Menus . . . . .	416
PopUp Menus . . . . .	424

---



## Chapter 22. Overview

The VisualAge for C++ Browser is a graphical tool for examining programs developed in VisualAge for C++.


This chapter briefly describes the Browser, its major features and concepts, and explains how you access the online contextual help and **How Do I?** information while you are using the Browser.


---

### Understanding the Browser



The graphical user interface consists of two types of Browser windows:

- A *List window* displays a list of program elements, such as source files, functions, and classes.
- A *Graph window* displays program relationships in a graphical format, for example, inheritance in a class hierarchy. You can specify the level of detail you want the graph to show, scroll over the graph, zoom in and out, and select program elements directly from the graph.


 For more information on List and Graph windows or other elements of the Browser user interface, see Chapter 24, “Understanding and Using the Browser User Interface” on page 339.


You can view your program files (.DLL, .EXE, .LIB) and compiler generated Browser (.PDB) files. In addition, if you have a IBM WorkFrame project, you can view your programs, without recompiling, by using the Browser QuickBrowse facility.  See “Using QuickBrowse” on page 389 for more information on QuickBrowsing your programs.

With the Browser, you can look at your source code in many different ways:

- List program objects by type (for example, all classes), by content (members of a class), or by components (all files).  See “Ordering the Contents of a Container View” on page 342 for more details.
- View relationships between program components graphically, such as class-inheritance hierarchies, function calls, and included files.  See “Using the Browser to Aid Program Understanding” on page 378 for more information.
- View and edit the actual source code associated with a selected program element using an editor. By default, the Browser uses the VisualAge Editor, but you can select to use a different editor by changing your IBM WorkFrame project

## Browser: Concepts

settings.  For more information on viewing and editing with the Browser, see “Editing and Viewing Source Files” on page 375.

- View online documentation for a class or class member.  For more information on viewing online documentation through the Browser, see “Showing VisualAge for C++ Open Class Library Documentation” on page 377.

---


## Concepts Used by the Browser




**Browsing** Browsing is a navigational technique for understanding the complexities inherent in a set of classes. The VisualAge for C++ Browser helps you navigate through an inheritance hierarchy, to understand the full interface of a class available to you as a programmer, to locate the body of a function or a class definition amongst dozens (possibly hundreds) of files across multiple directories, to understand calling relationships between functions, and much more.




**Browser Database** The Browser does not use an external database, but has an internal representation of your program. This representation reflects all the needed information for the actions that the Browser performs. This database can be populated from .PDB files (the richest source of information) or directly from source via the QuickBrowse facility in the Browser. .PDB files are generated when you run the compiler with the /Fb option. In addition, you can populate the Browser database from .DLL or .EXE files when you have also run the linker with the /BROWSE option, or from .LIB files that were built by the **ilibr** utility without the /NOBROWSE option. The Browser will save the contents of these files upon exit to a .PDD, .PDE, or .PDL Browser database file.

When you next load one of the program files, the Browser will use the stored Browser database file, updating it if you have modified your program since the last load. The Browser database files make loading your programs quicker.  For more information on Browser database files, see “Creating Files to Use with the Browser” on page 335. For more information on QuickBrowse, see “Using QuickBrowse” on page 389.




**Container View** A container view is a list which can be viewed using the + and - icons to expand and collapse the entries.  For more information on this type of view, see “Types of List Windows” on page 341.



**Objects** The Browser presents program elements to you in the form of objects on the screen. Objects can be classes, types, functions, variables, and source files. Use Mouse Button 2 on any of these objects to invoke a PopUp menu of actions available for that object.  For detailed descriptions of the various objects, see “Browsing List Objects” on page 343.





**Object-Action Pairs** You can perform actions on any Browser object from the object's PopUp menu. Use the Mouse Button 2 over the object to invoke the PopUp menus. The Browser remembers the last 40 object-action pairs and lists them in the **History** window for quick access.  “The History Window” on page 373 gives you more information on using the **History** window.

The List and Graph window each have an **Action Status Bar** which is located directly below the menubar. It indicates what object and action were used to create the current contents of the window.

---


## Using the Mouse



The Browser makes use of both Mouse Button 1 (usually the left mouse button) and Mouse Button 2 (usually the right mouse button).

Use **Mouse Button 1** for selecting objects in the windows, for scanning the Browser PullDown menus, and for sizing windows and selection areas.

Use **Mouse Button 2** for performing actions on the various objects displayed in the windows. Each type of object has several associated actions in the form of a PopUp menu.

 See “Browsing List Objects” on page 343 for more information on the types of objects available through the Browser.

---

## Getting Help While You Are Using the Browser

In addition to this guide, there is:

- Contextual online help, which gives you help from within the Browser.
- **How Do I?** help, which gives you step-by-step instructions on how to perform several Browser related tasks.

### Using Contextual Help

You can ask for help on any menu choice, window, or entry field on how to use a particular item by accessing the online context-sensitive help. You can access it in one of the following ways:

- Select a choice from the Help PullDown menu.
- Press **F1** from any Browser window.
- Press **F1** while highlighting any menubar item.
- Press **F1** while highlighting any PullDown menu item.
- Press **F1** or select the **Help** PushButton on any dialog or NoteBook.

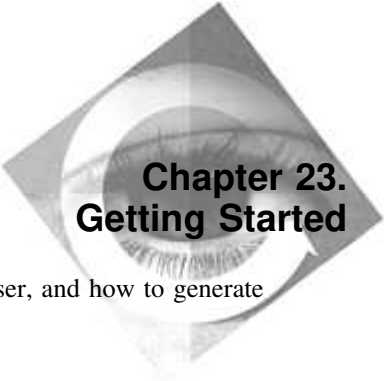
## Browser: Getting Help

### Using the How Do I... Information

The **How Do I?** information can help you quickly accomplish tasks when you are unclear on what to do next.

You can access the **How Do I?** information in a number of ways:

- Select the Help PullDown menu and select the How Do I... menu item from the Browser.
- Select the Help PullDown menu from any component of VisualAge for C++, select the **How Do I... Selections** ► Cascade menu, and select the **Browser** menu item.
- Open the **Information** folder located in the main VisualAge for C++ folder on your desktop. Open the **How Do I?** folder and select the **Browser How Do I?** information.



## Chapter 23. Getting Started

This chapter tells you how to start and close the Browser, and how to generate Browser database files.

---

### Starting the Browser

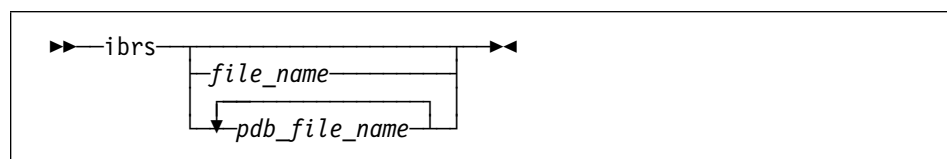
You can start the Browser from four different places:

- The command line.
- The Program Manager.
- The IBM WorkFrame environment.
- The VisualAge for C++ Debugger.

If the program you want to browse is large, the Browser may take several seconds to load. A **Progress** dialog will appear to inform you of the progress. You can see this dialog in Figure 109 on page 369.


### From the command line

To start a Browser session from the command line, use the **ibrs** command as follows:



Where:

*file\_name* Can be a Browser database file (with extension .PDB, .PDE, .PDD, .PDL) or a program file (with extension .EXE, .DLL, .LIB).


*pdb\_file\_name* Can be multiple .PDB file names. You can load multiple .PDB files into the Browser. You cannot load any other combination of files into the Browser, however, you can merge multiple file types.  See “Merging Files” on page 369.

If you type **ibrs** without any options, the Browser starts without any loaded files. From the File PullDown menu:

- Select the Load... menu item in order to launch the **Load Database** dialog. From this dialog, you can enter a program file name with extension .DLL, .EXE or .LIB, or a Browser database file with extension .PDB, .PDD, .PDE, or .PDL, or


## Browser: Getting Started


- Select one of the libraries that make up the VisualAge for C++ Open Class Library directly from the Load ► Cascade menu found on the File PullDown menu.

 For more information on loading, see “Loading Files into the Browser” on page 368.

## From the Program Manager


You can start the Browser from the Program Manager in three ways:

- Double-click on the Browser icon. You can load a database file name or a program file name from the **Load Database** dialog which you can invoke from the File PullDown menu.  For more information on loading, see “Loading Files into the Browser” on page 368.
- Double-click on a Browser database file (.PDB, .PDD, .PDE, or .PDL) from WorkFrame folder.
- Drag a Browser database file (.PDB, .PDD, .PDE, or .PDL) or program file (.DLL, .EXE, or .LIB) or IBM WorkFrame project icon onto the Browser icon.

 For a description of a Browser database, see “Concepts Used by the Browser” on page 330. For information on creating database files, see “Creating Files to Use with the Browser” on page 335.

## From the IBM WorkFrame environment

To start the Browser from the IBM WorkFrame environment, you can double-click on a .PDB, .PDE, .PDD, or .PDL Browser database object in any WorkFrame project folder, or you can select **Browse** from any **WorkFrame Project** PopUp menu to load a Browser session. You can also select the **Browse** action on any .PDB, .PDD, .PDE, .PDL, .EXE, .DLL, or .LIB file.

 For information on creating database files, see “Creating Files to Use with the Browser” on page 335. For more information on creating a project, setting options, and starting tools from the WorkFrame environment, see Part 1, “Developing with WorkFrame” on page 1.


## From the VisualAge for C++ Debugger

To start the Browser from the Debugger:

1. Select the Project PullDown menu from the Debugger user interface.
2. Select the **Browse** Cascade menu.
3. Select a Browser action to perform on the program currently loaded into the Debugger or on a selected program element.

### Creating Files to Use with the Browser



The Browser can view either Browser database files (.PDB, .PDD, .PDE, .PDL) or programs files (.DLL, .EXE, .LIB). The following describe how to compile and link your source files, and how the Browser generates Browser database files from your loaded source.  For a description of what a Browser database file is, see “Concepts Used by the Browser” on page 330.

#### Created by Compiling

The compiler will generate .PDB Browser database files (one per compilation unit), if you compile your source with the /Fb option. This provides the Browser with the richest source of information. However, there are two options that you can use to compile your programs for use with the Browser: **/Fb** or **/Fb\***.

The difference between the two options relates to how much Browser information is generated from system include files. That is, those included via:

```
#include <system_file.h> as opposed to:
#include "local_file.h".
```

The Generate Browser information option (**/Fb**) discards much of the non-type information from inside of system include files. For instance:

- Non-member function declarations will not be included in the .PDB file, including those C and system header files. Any friendship granted to these types of omitted functions will not be recorded for a class. For instance:

```
// in <foo.h>
int foobar(void);
// in "bar.h"
class bar {
    friend int foobar(void);
}
```

This friendship will not be included in the list of friends of class bar.

- No global variable declared, or defined, in the system header file will be included in the .PDB file. This includes variables of an instantiated template type.
- Class member function declarations will be added to the .PDB file, but their inline definitions, if any, will not.
- Non-inline function definitions will not be added to the .PDB file.

These restrictions are lifted when compiling with the Generate All Browser Information (**/Fb\***) option. It is recommended that you use the /Fb option, unless the compiler issues a message indicating that the use of the /Fb\* option is appropriate.

## Browser: Closing the Browser


### Created by Linking

In addition, you can populate the Browser database from .DLL and .EXE when you have run the linker with the /BROWSE option, and from .LIB files when you have run the **ilibr** utility without the /NOBROWSE option. Note that the linker will create additional .PDB files for template instantiations from TEMPINC.

### Created by the Browser

Upon exit, the Browser will save the contents of the .DLL, .EXE, and .LIB files to Browser database files. For example, you can load an .EXE file:

```
ibrs wombat.exe
```

or enter the program file name into the **Load Database** dialog.  For more information on loading, see “Loading Files into the Browser” on page 368. When you view a program file for the first time, the Browser generates a database file with an extension corresponding to the extension of the file you are viewing.

File Extension	Database File Extension
.EXE	.PDE
.DLL	.PDD
.LIB	.PDL


In the example given above, the Browser would generate the file wombat.pde.

Note that this single file contains all the information linked from every compilation unit's .PDB files, in the same way that the .EXE is the linked composite of all its compilation unit's .OBJ files.

If you invoke the Browser with a program file for which the Browser has already generated a Browser database file, the Browser uses that database file instead of generating a new file. If the file has been updated, the Browser updates the corresponding database file. This is called an “incremental load”. For example, the next time you view wombat.exe, the Browser uses the wombat.pde file that it created, thus reducing the time it takes to load the Browser with the wombat.exe information. If you have made changes to wombat.exe, these changes are loaded from the new .PDB files. The incremental load reduces the loading time of the new information.

---

## Closing the Browser

To end the Browser session, select Exit Browser from the File PullDown menu in any List or Graph window, or press **F3**. A dialog appears asking if you want to exit your current Browser session. Select **Yes** to exit or **No** to return to your Browser session. You can prevent being prompted by this dialog by selecting the Browser... menu item from the Options PullDown menu. Deselect the **Confirm on exit** CheckBox.  For more information on the **Browser Settings** Notebook, see “Changing Browser Settings” on page 363.

## **Browser: Closing the Browser**


When you exit the Browser, the currently loaded Browser database is saved to a Browser database file. By default, the Browser names these files based on the file name and adding either the .PDD, .PDE, or .PDL extension, depending on whether the file was a .DLL, an .EXE, or a .LIB, respectively. If the Browser cannot determine what the database file should be named, a message may prompt you to name your Browser database file. Also, if you performed a merge or loaded one or more .PDB file, the Browser will ask you to give the Browser database file a name.

## **Browser: Closing the Browser**





## Chapter 24. Understanding and Using the Browser User Interface

This chapter describes the Browser windows and dialogs. The descriptions cover what each window or dialog is for, and how you setup and use them.  For more details on using the Browser to understand and develop your applications, see Chapter 25, “Using the Browser” on page 375. For a quick tour of the Browser features, try Chapter 26, “A Tour of the Browser” on page 395.



The following are described in this chapter:

- “The List Window” on page 340.
- “The Graph Window” on page 351.
- “Changing Browser Settings” on page 363.
- “Changing Fonts” on page 366.
- “Loading Files into the Browser” on page 368.
- “Merging Files” on page 369.
- “Finding Objects in the Current Window” on page 371.
- “Searching for Objects in the Entire Browser Database” on page 372.
- “The History Window” on page 373.

## Browser: List Window

### The List Window



The List window displays a list of program elements such as classes, functions, and source files that were used to generate your program. Many aspects of a program can be described as a list. Other aspects can be described as relationships in a Graph window described on page 351.

To view your programs with the Browser, load your program files (.DLL, .EXE, or .LIB) into the Browser using the **Load Database** dialog described on page 368.

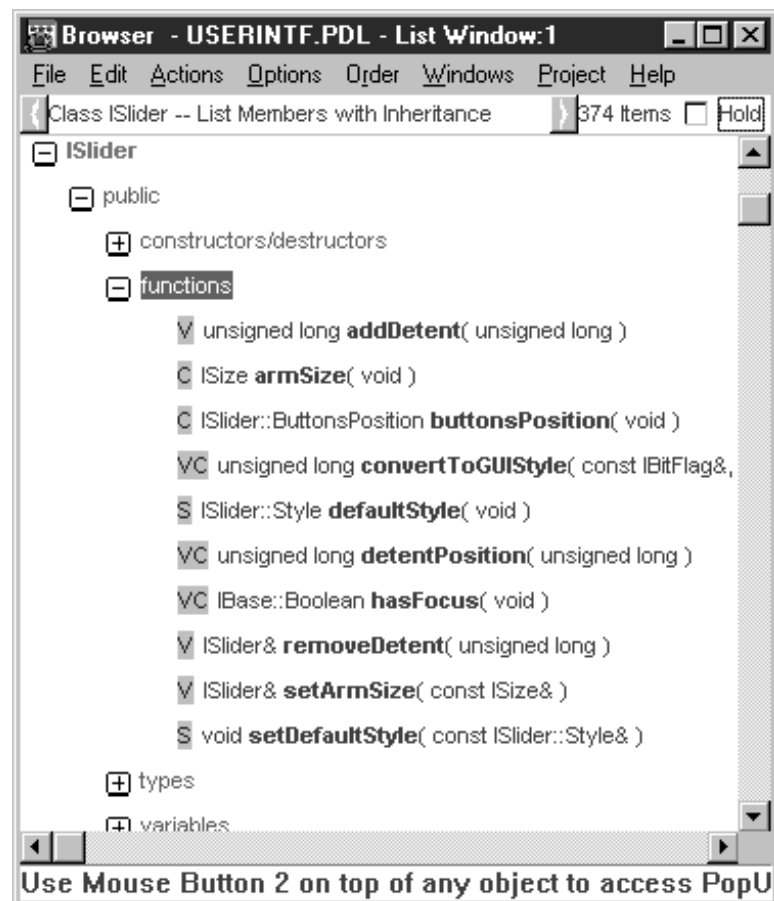






Figure 84. List window showing a container view

## Browser: List Window

The List window consists of:

- A main menu bar whose PullDown menu actions affect the current window.  See “PullDown Menus” on page 416 for detailed descriptions of the PullDown menus.
- An **Action Status Bar** which describes what object and action produced the currently viewed list. A definition of *object-action pair* is described on page 331. In the above figure, the **Action Status Bar** is “Class ISlider - List Members with Inheritance”.
- A count that indicates how many program elements are in the current list, excluding label objects. In the above figure, there are 367 program elements listed.
- A **Hold** CheckBox that, when checked, keeps the current List window contents from being replaced when an object-action pair results in a list. This can be useful, for instance, if you want to keep a list of all classes available at the same time that you want to focus on one given class. If the **Hold** CheckBox is not checked (the default behavior), the results of the next action overwrites the current window contents. There is a limit of four List windows.
- A **List Area** where your program objects are listed. You can perform actions on the contents of the **List Area** using Mouse Button 2 on any of the listed items or on the background of the List window. You can also print, save to a file, or copy to the clipboard the contents of the List window. In the above figure, the **List Area** is displaying a container view.  For a description of what a container view is, see “Types of List Windows.”
- An **Information Bar** at the bottom of the window quickly defines the currently selected menu item or explains how to invoke the Object PopUp menus. You can hide the **Information Bar** by selecting the **Expert** help level in the **Browser Settings** dialog.  See “Changing Browser Settings” on page 363 for more information.

The List window has a **Background** PopUp menu you use to perform actions on the window. You can access it using Mouse Button 2 on the background of the List window.  See “PopUp Menu Items for List and Graph Windows” on page 424 for detailed descriptions of the window PopUps.

### Types of List Windows



The first List window that you see after loading a program into the Browser displays all the classes defined in the loaded Browser database. This is referred to as a *straight view*. The Browser has one other type of List view, a *container view* to list:

- All members of a class and its base classes (List Members with Inheritance on a class or List Class Members with Inheritance on a function),
- All defined objects in a file (List Defined Objects on a file),
- All friends of a class (List Friends on a class), or

## Browser: List Window

- All immediate callers and callees of a function (List Immediate Callers & Callees on a function).

All container views are expandable. (See Figure 84 on page 340 for an example of a class and its base classes container view). You can use the + and - icons to expand and collapse the current selection one level, or use the **F7** and **F8** keys to expand and collapse the entire contents of the window.

### Ordering the Contents of a Container View



When you perform a List Members with Inheritance action on a class or a List Class Members with Inheritance on a function, the resultant list is a container view. You can rearrange the contents of this particular kind of container view. The container views that you get when you perform the List Defined Objects, List Immediate Callers & Callees, or List Friends actions only have one level of expansion, so it does not make sense to order them based on class, access or type.

You can order the contents of the container view resulting from a List Members with Inheritance or List Class Members with Inheritance action by:

**Class** Ordered by class, access, then type:

- Class 1
  - Public
    - + Constructors/Destructors
    - + Functions
    - + Types
    - + Variables
  - + Protected
  - + Private
- + Class 2

**Access** Ordered by access, type, then class:

- Public
  - + Constructors/Destructors
  - Functions
    - + Class 1
    - + Class 2
  - + Types
  - + Variables
- + Protected
- + Private

## Browser: Objects

**Type**     Ordered by type, access, then class:

- + Constructors/Destructors
- Functions
  - + Public
  - Protected
    - + Class 1
    - + Class 2
  - + Private
- + Types
- + Variables

By default, the List window displays a container view with the classes at the highest level (the Class order). If you are interested in seeing particular program elements (like all functions used by your program) from various classes or particular access methods (like what is defined as public), it can become quite annoying to have to scroll back and forth in the list between the classes. The Browser eliminates this annoyance by giving you the ability to reorder the list contents. You can view all the program elements together by type or together by access method, eliminating the need for constant scrolling of the list contents.

**Note:** The resultant list when you perform the List Members with Inheritance or List Class Members with Inheritance action is not sorted alphabetically as items in a List window generally are. The classes are arranged in a depth first tree traversal of the classes inheritance hierarchy.

### Browsing List Objects



The Browser presents program elements to you in the form of objects on the screen. Objects can be classes, types, functions, variables, and source files. In addition, the Browser uses Label objects to organize the program elements on the screen. You can click Mouse Button 2 on any of these objects to invoke a PopUp menu of actions available for that object.

#### Class Objects

C++ classes, structs, unions, class templates, class template instantiations, classes defined at file (or global) scope, and those nested inside of other classes. Classes defined inside of function bodies are not included.

Class objects can be specialized as SOM classes. A SOM class is any class that derives from SOMObject, and a SOM metaclass is any class derived from SOMClass. C++ Direct-To-SOM classes are supported by the Browser.

Bindings from the C++ SOM emitter may also be browsed, but the symbols produced by the emitter may not be easy to read or understand.

## Browser: Flags

### Function Objects

Functions, member functions, function templates, function template instantiations, and SOM methods. Class member functions defined inside of function bodies are not included.

### Variable Object

Variables, class member variables, class template member variables, member variables of class template instantiation, and SOM data members. Not included are variables defined inside of function bodies.

### Type Objects

Typedefs and enums. Not included are enumeration values, nor those enums and typedefs declared inside of function bodies, or built-in C types, such as int or char, or any pointer or reference combination of these types. Classes, structs, and unions are referred to as class objects.


### File Objects

Source files used to create your loaded program.

### Label Objects

When you perform either the List Members with Inheritance, List Friends, List Class Members with Inheritance, List Defined Objects, or List Immediate Callers & Callees action, the results are placed in a List window container view. In this type of view, Label objects are used to organize the results in the List window by category. Categories can be nested. The categories are Public, Protected, Private, Constructors/Destructors, Classes, Functions, Types, Variables, Callers, and Callees. Also, the results displayed when you expand a typedef are label objects.

To browse the program elements listed in the List window, use the Object PopUp menus. You can invoke them by selecting an object and clicking on Mouse Button 2.

 See "Object PopUp Menu Items" on page 425 for detailed descriptions of the Object PopUp menus.

## Understanding Browser Generated Flags



Sometimes you will see one of the following three flags when viewing items in the List window:

### [anonymous]

The C language has the concept of anonymous structs, unions and enums which C++ has inherited. For instance, each of the types below has no name (or is anonymous):

## Browser: List Window Settings

```
struct {  
    int number;  
    int code;  
    char *name;  
} record;  
union {  
    int a;  
    char b;  
};  
enum { red, yellow, green } color_1, color_2;
```

### [compiler generated]

When you define a class, but do not define your own default constructor, copy constructor, or destructor, the compiler will quietly generate one for you. We thought it important to let you know in the Browser when this occurs.

### ('n' instances)

When you use multiple inheritance and two (or more) of your base classes inherit from a common class such as the following:

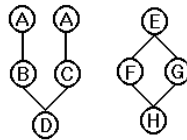



Figure 85. ('n' instances) example

Non-virtual inheritance was used by B and C when they inherited from A, but virtual inheritance was used by F and G when they inherited from E. An object of type D will contain two copies of the data contained in an object of type A. An object of type H will contain only one copy of the data contained in an object of type E.



To highlight this non-diamond shape inheritance structure to you, the Browser indicates when a derived class contains more than one copy of the data from a base class.

## Changing the Default List Window Settings

The Browser provides you with a **List Windows Settings** Notebook from which you can change:

**Settings** Used to change the **Action Status Bar** font, the initial action performed when you load a file into the Browser, and the double-click actions of the objects displayed in the List window.  See “Changing List Settings” on page 346.

## Browser: List Window Settings

- Colors** This page is used to change the colors of objects in the List window.  See “Changing Colors Used by the List Window” on page 347.
- Styles** This page is used to change the amount of text displayed by the List window.  See “Changing the List Style” on page 349.

### Changing List Settings

You can change the **Action Status Bar** font, the initial action performed when you load a file into the Browser, and the double-click actions of the objects displayed in the List window.

To load the **List Window Settings** Notebook, select List Window... from the Options PullDown menu.

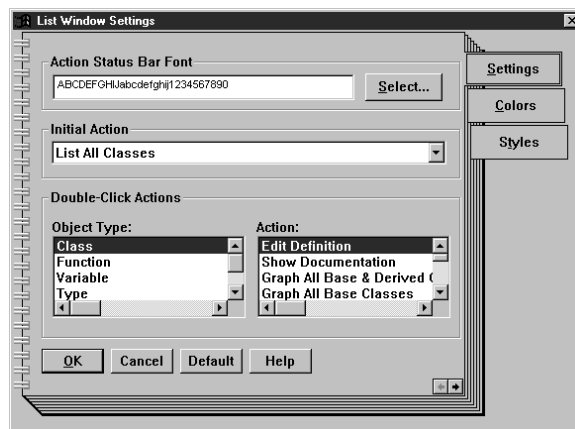



Figure 86. List Window Settings Notebook Settings Page

You can change the font used by the **Action Status Bar** independently of the font used in the **List Area** of the List window. To change the font of the **Action Status Bar** the List window, choose the **Select...** PushButton to load the **Action Status Bar Font** dialog.  For more information on the **Fonts** dialog, see “Changing Fonts” on page 366.


You can change the initial action performed when you load a program into the Browser. Use the **Initial Action** DropDown to select the action to perform when loading a program into the Browser from the List window. By default, the List window displays all the classes defined in your loaded program. You can choose from List All Classes, List All Files, or **None**.

Use the **Double-Click Actions** section to change the default action performed when you double-click on an object in the List window. By default, the double-click actions for all objects, except Labels, is to edit the object's definition. Select the



## Browser: List Window Settings

object from the **Object Type:** ListBox, and select a corresponding action from the **Actions:** ListBox.

**Note:** These settings are independent of the selections made for the Graph window settings.  See “Changing Graph Settings” on page 356.


### Changing Colors Used by the List Window

You can identify objects in the List window by their color. By default, the following colors are used:

<u>Object</u>	<u>Color</u>
<b>Class</b>	Cyan
<b>SOM Class</b>	Dark Green
<b>SOM Metaclass</b>	Light Green
<b>Function</b>	Blue
<b>Type</b>	Blue
<b>Variable</b>	Blue
<b>File</b>	Red
<b>Label</b>	Brown

In addition, the following letters, displayed in light blue by default, are used to further indicate the attribute of a function, type or variable:

<b>C</b>	Constant (functions)
<b>V</b>	Virtual (functions)
<b>E</b>	Enumerator (types)
<b>S</b>	Static (functions and variables)
<b>PV</b>	Pure Virtual (functions)

**Note:** You can expand the attributes into their complete name using the **List Window Settings** Notebook Styles page which is described in  “Changing the List Style” on page 349 .

You can customize the colors used in the List window because:

- You may prefer different color schemes.
- Different monitor resolutions, or color palettes may make the default Browser color choices indistinguishable.
- Changing the colors can make the types of objects that you are interested in stand out more and tone down those that you are not interested in.
- When printed, the results may look better when different colors or fonts are used. The results may be different for the List window and Graph window views, so you have the ability to change them for each type of window.

Note that making a color change will affect all open List windows and all subsequently opened List windows. The new defaults are saved to the Browser profile (ibrs.ini) when you exit from the Browser.

## Browser: List Window Settings

To change the colors used by the List window, select List Window... from the Options PullDown on a List window to start the **List Windows Settings** Notebook. Select the Colors tab.

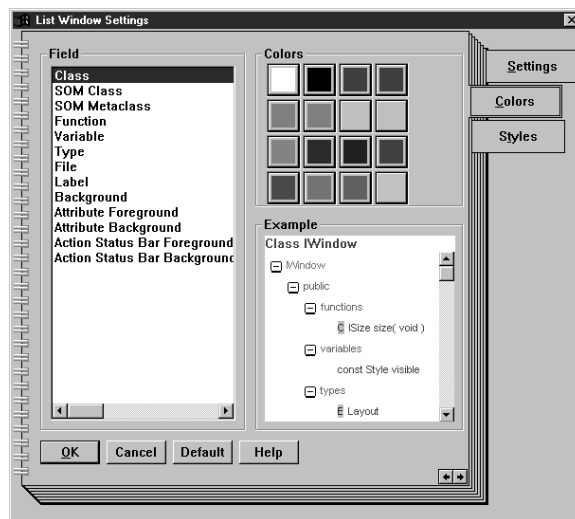


Figure 87. List Window Settings Notebook Colors Page

On the left side of the page is a scrollable list of items for which you can set colors. On the upper right side is the color palette containing 16 colors to choose from. On the bottom right side is an **Example Area** that will give you a preview of the color selection that you are currently editing.


To change an object's color:

1. Select the object from the scrollable list.
2. Click on the color from the 16 available colors. Your change appears in the **Example Area**.
3. Select the **OK** PushButton if you want to apply your color changes to the List Window, or select the **Cancel** PushButton if you want to exit without making any changes.

The **Default** PushButton restores the default colors used by the Browser List windows. In addition to the default colors used by the objects in the List window (mentioned above), you can change:

## Browser: List Window Settings

<u>Window Attribute</u>	<u>Color</u>
<b>Background</b>	White
<b>Attribute Foreground</b>	Dark Red
<b>Attribute Background</b>	Light Blue
<b>Action Status Bar Foreground</b>	Blue
<b>Action Status Bar Background</b>	White

**Note:** These settings are independent of the selections made for the Graph window color settings.  See “Changing Colors Used by the Graph Window” on page 358.

### Changing the List Style

You can change the amount of text that is displayed in the List window. To change the List window text style, select List Window... from the Options PullDown menu to start the **List Window Settings** NoteBook. Select the **Styles** tab.

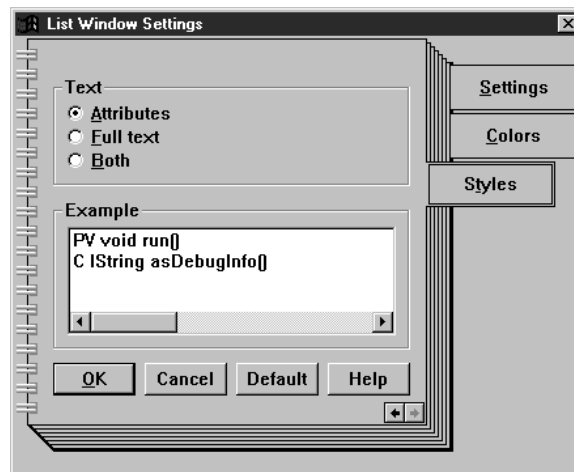



Figure 88. List Window Settings NoteBook Style Page

There are three text style settings:

- Attributes** Summarizes the program element attributes (C-constant, V-virtual, E-enumerator, S-static, and PV-pure virtual).
- Full Text** Lists the full text of the program element attributes.
- Both** Lists both the summary and full text of the program element attributes.

The **Example Area** shows what the List window text will look like if you choose the different options. Select **OK** to accept the changes and **Cancel** to exit without making changes.

## Browser: Printing and Saving Lists

Note that these settings are independent of the selections made for the Graph window style settings.  See “Changing Graph Styles” on page 359.

### Printing and Saving your Lists

You can print the currently displayed list using Print... from the File PullDown menu. The print job will use as many pages as required to print the entire list, therefore the **Multiple Pages - Grid Layout** section of this dialog is disabled. Select the **Print** PushButton to print the current list. Select the **Print Setup...** PushButton to change the printer properties and page setup. Select the **Fonts...** PushButton to change the fonts used when printing. Note that the name of this dialog includes the type of printing you are requesting. In this case, you are requesting to print a list.

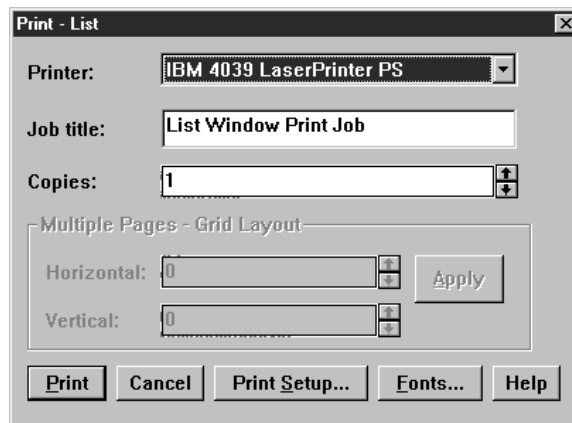


Figure 89. List Print Dialog

You can also save the list as an ASCII file. Note that the colors are not saved to the ASCII file. Select the drive and directory that you want to save the list to, and enter a filename into the **Save as filename:** TextEntry field, or enter the drive, path name, and file name into the TextEntry field.

## Browser: Graph Window

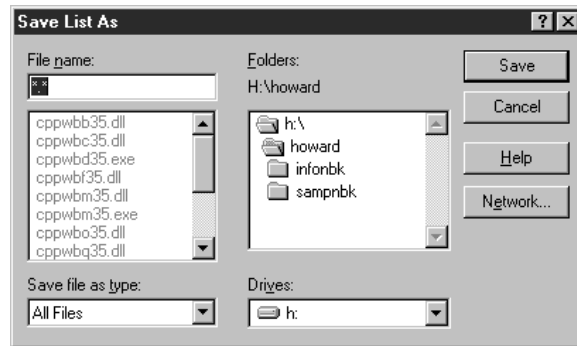



Figure 90. Save As... Dialog


---

## The Graph Window



A Graph window displays program relationships in a graphical format. You can specify the level of detail you want the graph to show, scroll over the graph, zoom in and out, and select program elements.

Many aspects of a program can be described as relationships between program elements.  Other aspects can be described by listing some group of elements in a List window described on page 340.

To use the Graph window to view your program relationships, load your source files into the Browser using the  **Load Database** dialog described on page 368.

## Browser: Graph Window

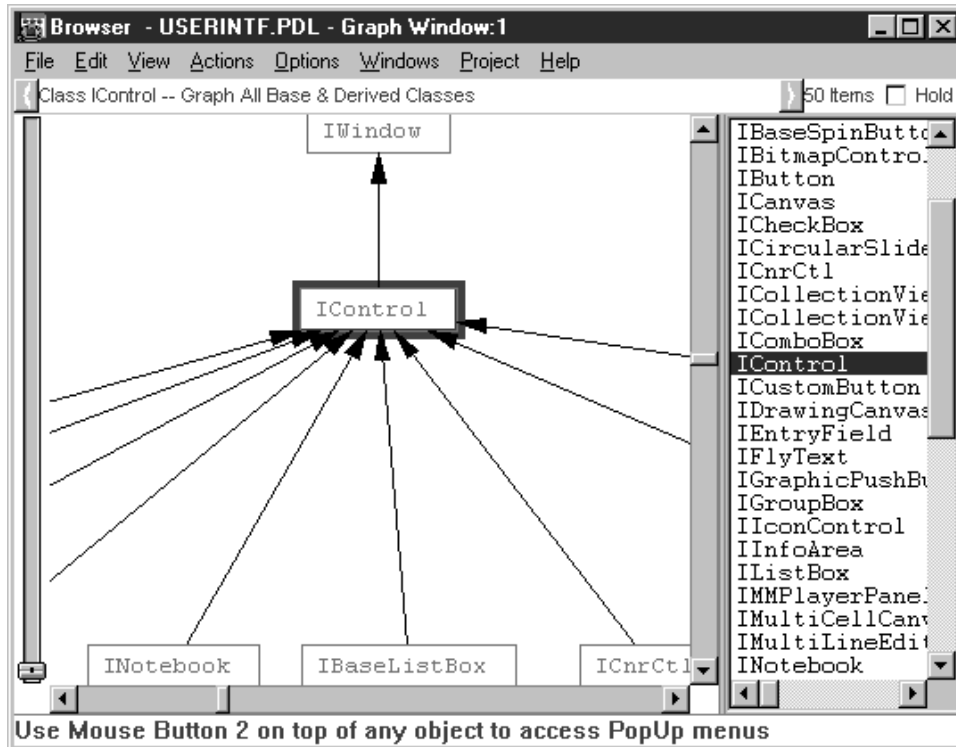



Figure 91. Graph Window


The Graph window consists of:


- A main menu bar whose PullDown menu actions affect the current window. For detailed descriptions of the PullDown menus, see [“PullDown Menus”](#) on page 416
- An **Action Status Bar** which describes what object and action produced the current graph. A definition of *object-action pair* is described on page 331. In the above figure, the **Action Status Bar** is “Class IControl - Graph All Base & Derived Classes”.
- A count that indicates how many program elements (nodes) are in the current graph. In the above figure, there are 49 class nodes.
- A **Hold** CheckBox that, when checked, keeps the current Graph window contents from being replaced when an object-action pair results in a graph. This can be useful, for instance, if you want to keep an inheritance graph available at the same time that you want to focus on another relationship. If the **Hold** CheckBox is not checked (the default behavior), the results of the next action overwrites the current window contents. There is a limit of four Graph windows.

## Browser: Graph Window

- A **Graph Area** that contains the graphical results of your object-action pair. You can perform actions on the contents of the **Graph Area** using Mouse Button 2 on any of the nodes or on the background of the Graph window. You can also print, save to a file, or copy to the clipboard the contents of the **Graph Area**. When you select a node on the graph, the selected item is highlighted in the **List Area** of the Graph window.
- A **List Area** that alphabetically lists all the nodes on the graph. You can click on the items in this list to see where in the graph the node appears. This node is highlighted in red, by default. You can perform actions on the contents of the **List Area** using Mouse Button 2 on any of the listed items or on the background of the **List Area**. You can also save the contents of the **List Area** to an ASCII file.
- A **Slider** on the left side to quickly zoom in and out on the graph. Move the **Slider** up to reduce the size of the graph and down to increase the size.
- Scroll bars located on the right side and bottom of the **Graph Area**. Use these to scroll the graph horizontally and vertically.
- A divider located between the **Graph Area** and **List Area**. Use it to change the proportion of the screen allocated to each area.
- An **Information Bar** located at the bottom of the window that briefly describes the currently selected menu item or explains how to invoke the Object PopUp menus. You can hide the **Information Bar** by selecting the **Expert Help** level in the **Browser Settings** dialog.  See “Changing Browser Settings” on page 363 for more information.

A very large number of nodes may exhaust the system resources. Note that any graph approaching this limit would not be clearly understandable in the Graph window.

The **Graph Area** has a **Background** PopUp menu you can use to perform actions on the window, such as, scrolling over the graph, zooming in and out, and changing the layout parameters. You can access this PopUp using Mouse Button 2 on the background of the **Graph Area**.  See “PopUp Menu Items for List and Graph Windows” on page 424 for detailed descriptions of the window PopUps.

You can select a portion of the **Graph Area** by clicking and dragging the mouse over the area that you want to select. You can then get a PopUp menu specific for the selected region that allows you to zoom in on, copy, save, or print the selected area.  See “Selecting a Graph Zone” on page 355.

## Browser: Graph Overview • Browser: Organizing a Graph

### Getting a Graph Overview



You can view an overview of the graph using the Overview... item on either the View PullDown or the Graph Window **Background** PopUp menu.

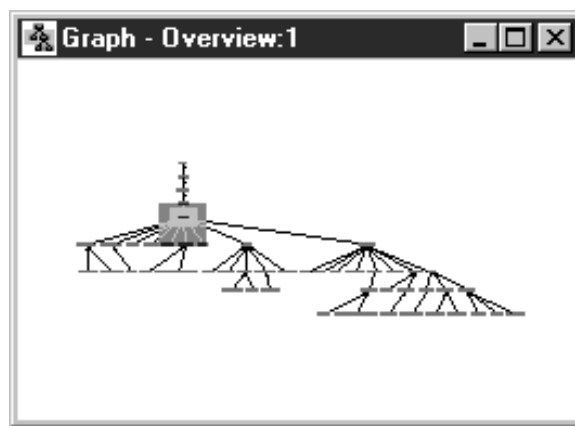


Figure 92. Overview Window

The grey shaded area indicates the current view of the **Graph Area** in the Graph window. You can move this area around or resize it. Any changes you make to the size or position of this grey area is automatically reflected in the Graph window.

Use  $\leftrightarrow$  and  $\updownarrow$  to size the view of the graph. The result is the same as if you had used the **Slider** on the left side of the Graph window. Use the four-way cross-arrow to move the grey area around. The result is the same as if you had use the scroll bars around the **Graph Area** of the Graph window.

**Note:** If you change the view of the **Graph Area** in the Graph window, this change is automatically reflected in the **Overview** window.

### Organizing the Graph

You can organize the graph using two kinds of layouts that work in conjunction with one another. By default, the Graph window displays graphs in a vertical layout; that is, the nodes are drawn from top to bottom. You can change this to a horizontal layout which redraws the nodes from left to right. For instance, some wide graphs with wide nodes look best when shown horizontally, while narrower graphs look best when shown vertically. To choose a vertical or horizontal layout:

1. Select either the View PullDown menu from the main menu bar, or select the Graph window **Background** PopUp menu by clicking Mouse Button 2 on the background of the **Graph Area**.
2. Select the Horizontal or Vertical menu item.



## Browser: Graph Zone

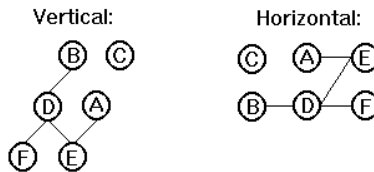


Figure 93. Horizontal versus Vertical Graph Layout

In addition to the horizontal or vertical layout, you can also weight the nodes of your graph. Some graphs are easier to understand when all root nodes or leaf nodes are at an equal level, or something somewhere in between. To select a different weighting:

1. Select either the View PullDown menu from the main menu bar, or select the Graph window **Background** PopUp menu by clicking Mouse Button 2 on the background of the **Graph Area**.
2. Select the Weighting ► Cascade menu.
3. Select the Top, Center, or Bottom menu item.

**Note:** If you have chosen a horizontal layout, then the root nodes will be grouped to the left and the leaf nodes will be grouped to the right.

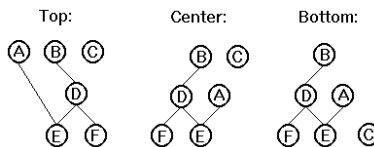


Figure 94. Top, Center and Bottom Weighting of a Graph

## Selecting a Graph Zone



You can select a particular region of the graph by clicking Mouse Button 1 and dragging it across the graph. This creates a rectangular dotted box around the selected region. You can get a **Graph Zone** PopUp menu using Mouse Button 2 on this region. This PopUp has the following actions:

- Zoom in** Zooms in on the selected region.
- Save Graph As...** Saves the selected region to a bitmap file.
- Print...** Prints the selected region.
- Copy All** Copies the selected region to the clipboard.

## Browser: Graph Window Settings

### Browsing Graph Objects



In the Graph window, you can display class, function and file objects. See “Browsing List Objects” on page 343 for descriptions of how the Browser defines these objects.

You can launch actions from either the nodes on the graph, or their corresponding list item in the **List Area** of the Graph window, by selecting the object and clicking on Mouse Button 2 to invoke the Object PopUp menu. For more information on Object PopUp menus, see “Object PopUp Menu Items” on page 425.

### Changing the Default Graph Window Settings

The Browser provides you with a **Graph Windows Settings** Notebook from which you can change:

- Settings** Used to change the **Action Status Bar** font, the initial action performed when you load a file into the Browser, and the double-click actions of the objects displayed in the Graph window. See “Changing Graph Settings.”
- Colors** Used to change the colors of objects in the Graph window. See “Changing Colors Used by the Graph Window” on page 358.
- Styles** Used to change the shape of the nodes and arcs in the Graph window. See “Changing Graph Styles” on page 359.
- Bitmap** Used to change the dimensions of a saved bitmap. See “Changing Bitmap Dimensions” on page 360.

### Changing Graph Settings

You can change the **Action Status Bar** font, the initial action performed when you load a file into the Browser, and the double-click actions of the objects displayed in the Graph window.

To load the **Graph Window Settings** Notebook, select Graph Window... from the Options PullDown menu.

## Browser: Graph Window Settings

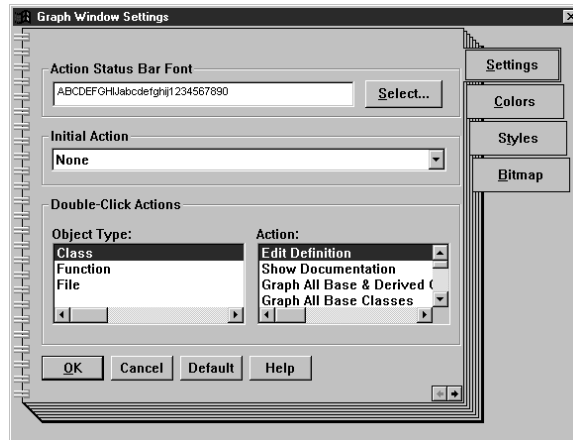



Figure 95. Graph Window Settings Notebook Settings Page

You can change the font used by the **Action Status Bar** independently of the font used in the **Graph Area** and **List Area** of the Graph window. To change the font of the **Action Status Bar** on the Graph window, choose the **Select...** PushButton to load the **Action Status Bar Font** dialog.  For more information on the **Fonts** dialog, see “Changing Fonts” on page 366.

You can change the initial action performed when you load a program into the Browser. Note that this action is only performed if you load a program while having a Graph window open. Use the **Initial Action** DropDown to select the action to perform when loading a program into the Browser. The Graph window does not have a default load action. You can choose from Show Inheritance Graph, Show Include File Graph, or **None**.

Use the **Double-Click Actions** section to change the double-click action of the objects in the Graph window. By default, the double-click actions for all objects is to edit the object's definition. Select the object from the **Object Types:** ListBox, and select a corresponding action from the **Actions:** ListBox.

## Browser: Graph Window Settings

### Changing Colors Used by the Graph Window

When you invoke an action to create a graph, the object you used to launch the action is highlighted in the Graph window in red, by default.

The Graph window displays each type of program element as a different color/shape and each relationship as a different colored/shaped arc.

Making a color change affects all open Graph windows and all subsequently opened Graph windows. The new defaults are saved to the Browser profile (ibrs.ini) when you exit from the Browser.

To change the colors used by the Graph window, select Graph Window... from the Options PullDown on any Graph window to get the **Graph Windows Settings** Notebook. Select the Colors tab.

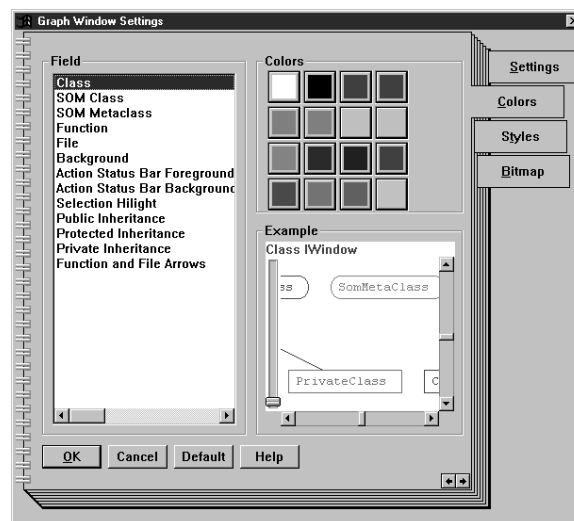


Figure 96. Graph Window Settings Notebook Colors Page

On the left is a scrollable list of items for which you can set default colors. On the upper right is the color palette containing 16 colors to choose from. On the bottom right is an **Example Area** that previews the color selection you are currently editing.

To change an object's color:


1. Select the object from the scrollable list.
2. Click on a color from the 16 available colors. Your change appears in the **Example Area**.

## Browser: Graph Window Settings

3. Select the **OK** PushButton if you want to apply your color changes to the Graph Window, or select the **Cancel** PushButton if you want to exit without making any changes.

The **Default** PushButton restores the default colors used by the Browser Graph windows. By default, the color settings are:

<u>Object</u>	<u>Color</u>
<b>Class</b>	Cyan
<b>SOM Class</b>	Dark Green
<b>SOM Metaclass</b>	Light Green
<b>Function</b>	Blue
<b>File</b>	Red
<b>Background</b>	White
<b>Action Status Bar Foreground</b>	Blue
<b>Action Status Bar Background</b>	White
<b>Selection Hilight</b>	Red
<b>Public Inheritance</b>	Black
<b>Protected Inheritance</b>	Red
<b>Private Inheritance</b>	Blue
<b>Function and File Arrows</b>	Black

**Note:** These settings are independent of the selections made for the List Window Style settings.  See “Changing Colors Used by the List Window” on page 347.

### Changing Graph Styles

You can change the shape of the nodes and the line style of the arcs using the Styles page of the **Graph Window Settings** Notebook.

## Browser: Graph Window Settings

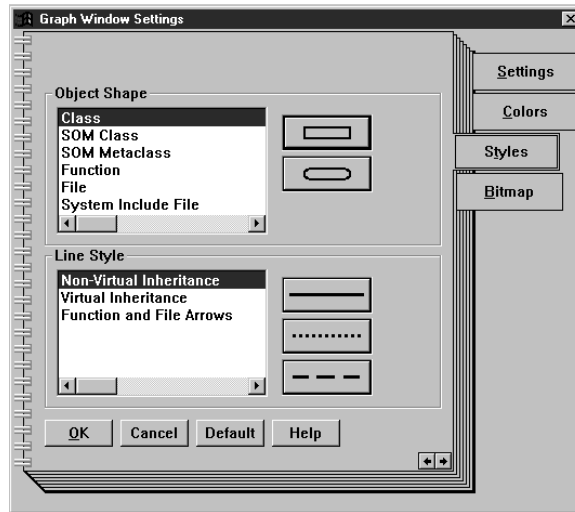


Figure 97. Graph Window Settings Notebook Styles Page

To change the shape of the nodes and arcs:

1. Select Graph Window... from the Options PullDown on the Graph window.
2. Select the Styles tab on the **Graph Window Settings** Notebook.
3. Select the object/relationship from the **Object Shape/Line Style** ListBox.
4. Select a shape/line PushButton.
5. Select **OK** to apply the changes to the Graph window, or select **Cancel** to exit without making changes.

These changes will be saved to the `ibrs.ini` profile when you exit the Browser.

The **Default** PushButton restores the Browser defaults for node shape and line style.

### Changing Bitmap Dimensions

You can specify the dimensions of the bitmap to be saved to a file or copied to the clipboard. Select Graph Window... from the Options PullDown menu on the Graph window. Choose the Bitmap tab on the **Graph Window Settings** Notebook. Enter the width and height of the bitmap that you want to save.

If you specify values for the width and height of the bitmap that do not correspond with the dimensions of the current graph or selection area, the output may not be as expected. For example, if the area to be saved or copied is a square and the values set specify a rectangular shape, then the image saved or copied will be stretched to fit the rectangle.

## Browser: Printing and Saving Graphs

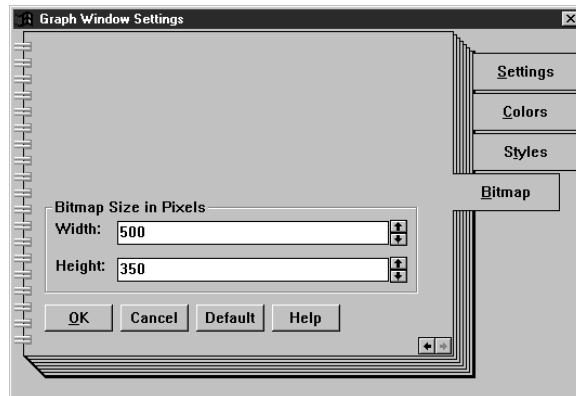


Figure 98. Graph Window Settings Notebook Bitmap Page

### Printing and Saving your Graphs


You can print the whole graph on one or several pages, print marked sections of the graph on one page, or print the currently viewed section of the graph on one page.

Use the Print ► Cascade on the File PullDown menu to print:

**One Page...** Print the entire graph on one page.

**Multiple Pages...** Print the entire graph across multiple pages.

**Client...** Print the currently displayed portion of the graph to one page.

**Zone...** Print the currently selected region to one page.  See “Selecting a Graph Zone” on page 355 for information on selecting areas of the graph.

By selecting any of the above print options, a **Browser Print** dialog appears. Note that the name of this dialog includes the type of printing you are requesting. In the example diagram below, you are requesting to print multiple pages.

## Browser: Printing and Saving Graphs

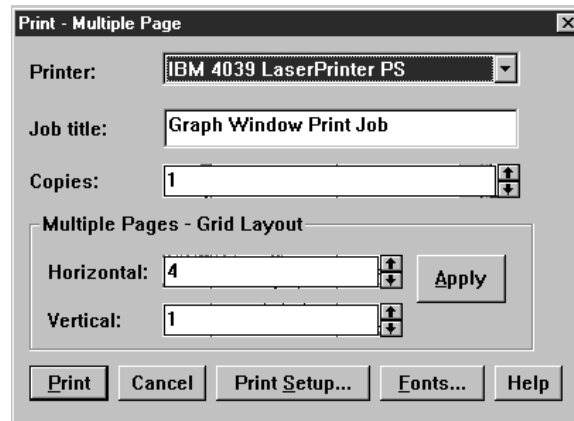



Figure 99. Graph Print Dialog

The **Multiple Pages - Grid Layout** section of the dialog is disabled when you choose to print One Page..., Client..., or Zone... from the Print ► Cascade menu on the Graph window. If you choose to print Multiple Pages... from the Print ► Cascade menu on the Graph window, you can select the layout of the pages to be printed by entering the number of horizontal and vertical pages to print. The Graph window zooms out to its maximum size, and the page layout is indicated by rectangular boxes on the graph. Each rectangular area is a page to be printed. Change the horizontal and vertical page numbers, and select the **Apply** PushButton to accept the new page layout. Selecting the **Apply** PushButton will redraw the grid layout on the graph.

Select the **Print** PushButton to print the current graph. If you chose to print multiple pages, then when the graph is finished spooling to the printer, the grid is cleared and the graph is restored to its previous zoom setting. Select the **Print Setup...** PushButton to change the printer properties and page setup. Select the **Fonts...** PushButton to change the font used when printing.

You can save either the **Graph Area** or **List Area** contents of the Graph window.

- Select Save Graph As... from the File PullDown menu on the Graph window to save the graph to a bitmap file.  See “Changing Bitmap Dimensions” on page 360 for more information on setting the size of the bitmap saved. Note that colors, shapes and layout of the graph are saved to the bitmap file.
- Select Save List As... from the File PullDown menu on the Graph window to save the list to an ASCII file.



## Browser: Path and Help Settings

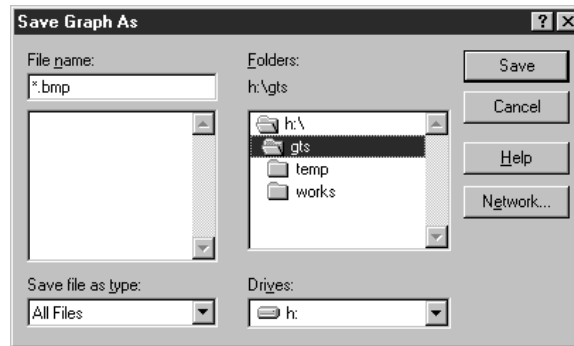


Figure 100. Save Graph As Dialog

---

## Changing Browser Settings

The Browser provides you with a **Browser Settings** NoteBook from which you can set:

- Paths** Used to change the file search path, the library files to be ignored by the Browser, and the directory in which you want to save the Browser profile (ibrs.ini). See “Changing Paths Used by the Browser” for more information.
- Help Level** Used to set the help level provided by the Browser, as well as allow you to disable the **Exit Browser** dialog. See “Changing Help Levels” on page 365 for more information.

## Changing Paths Used by the Browser

The Browser searches for files using the following sequence:

1. The path that was used to create the program.
2. The current directory.
3. The directories listed in the **Browser Settings** NoteBook Paths page.
4. The INCLUDE environment variable.
5. The DPATH environment variable.

Use the **Browser Settings** Paths page to specify the path names that the Browser should use in searching for Browser files and library files. You can also specify where to store the Browser profile (ibrs.ini).

**Note:** When the Browser lists files, it displays the path name of the files when the program was compiled. However, this path name may not be correct, as is the case with the Browser shipped .PDL files for the IBM VisualAge for C++

## Browser: Path and Help Settings

Open Class Library classes that you can load or merge from the Load ► and Merge ► Cascade menus.

To change the paths used by the Browser, select Browser... from the Options PullDown menu to start the **Browser Settings** NoteBook. The first page in this NoteBook is the Paths page.

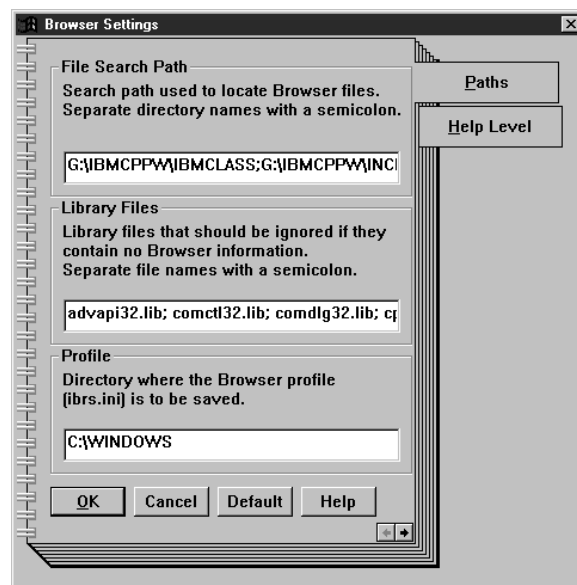


Figure 101. Browser Settings NoteBook Paths Page

Use the **File Search Path** TextEntry field to enter directory names where you want the Browser to search for files. By default, the path name contains the INCLUDE environment variable of the IBM VisualAge for C++ Open Class Library. Add your own directories to this list in the order in which the search should be performed. Place a semicolon (;) after each directory. If two directory names in the search path contain a file with the same file name, the file in the first directory will be used.

Use the **Library Files** TextEntry field to enter the names of the library files (.LIB) that your program uses, but where you do not want to see the Browser information for those files. Such files are from another vendor or files whose internals are not important to you. The files you specify here will most likely be those that you know do not contain any relevant Browser information. By default, the IBM VisualAge for C++ Open Class Library files are added to this list because they do not contain any Browser information. These library files are not used when the Browser loads all relevant information pertaining to your program or library. Place a semicolon (;) after each file name.

## Browser: Path and Help Settings

You may prefer, or need to, keep the Browser profile (ibrs.ini) in a location different from the one you specified when you first used it. You can use the **Profile** TextEntry field to change its location. When you exit the Browser, it will create the profile in this new location.

Select the **OK** PushButton to accept the changed path names. The new settings are saved to the Browser profile (ibrs.ini) when you exit from the Browser. Select the **Cancel** PushButton to exit without changing the path names. Use the **Default** PushButton to reset the path and file names to the Browser defaults.

### Changing Help Levels

Use the **Browser Settings** NoteBook Help Level page to specify different levels of help. Select Browser... from the Options PullDown menu to start the **Browser Settings** NoteBook., and select the Help Level tab.



Figure 102. Browser Settings NoteBook Help Level Page

You can set the level of help that the Browser provides:

- |                     |                                                                 |
|---------------------|-----------------------------------------------------------------|
| <b>New user</b>     | The <b>New User Help</b> dialog and an <b>Information Bar</b> . |
| <b>Intermediate</b> | No <b>New User Help</b> dialog, but an <b>Information Bar</b> . |
| <b>Expert</b>       | No <b>New User Help</b> dialog and no <b>Information Bar</b> .  |

The **Information Bar** is located at the bottom of each Browser window and is used to describes the various menu items as you highlight them.

You can turn off the **Exit Browser** dialog that appears each time you close the Browser by deselecting the **Confirm on exit** CheckBox.

## Browser: New User Help • Browser: Changing Fonts

### New User Help Dialog

When you first start the Browser, you get both a List window and the **New User Help** dialog. The information in the dialog outlines the key features of the Browser.

You can disable this dialog by unchecking the **Show new user help on startup** CheckBox or by selecting the **Intermediate** or **Expert** help levels. You can see this dialog again after disabling it by selecting the **New User** help level from the Help Level page in the **Browser Settings** Notebook.

The following four figures show the contents of the **New User Help** dialog:

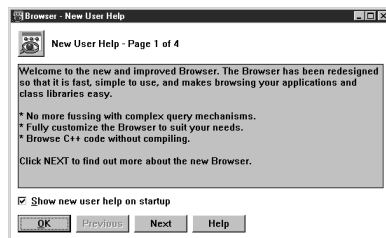


Figure 103. New User Help Dialog Page 1

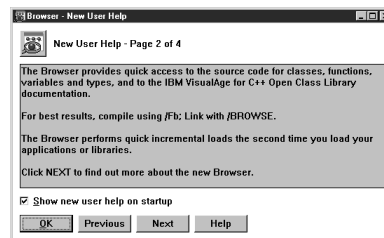


Figure 104. New User Help Dialog Page 2

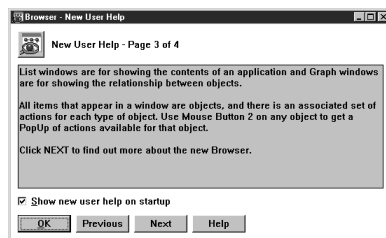


Figure 105. New User Help Dialog Page 3

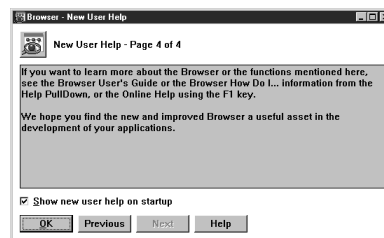


Figure 106. New User Help Dialog Page 4

---

## Changing Fonts

You can change fonts for the List window text, the List window **Action Status Bar**, the Graph window **Graph Area** text and **List Area** text, and the Graph window **Action Status Bar**.

## Browser: Changing Fonts

Change Font of:	How to:
List window text	Select Fonts... from the List window Options PullDown menu.
<b>Action Status Bar</b> in the List window	<ol style="list-style-type: none"> <li>1. Select List Window... from the Options PullDown menu to get the <b>List Window Settings</b> NoteBook.</li> <li>2. Select the Settings tab.</li> <li>3. Choose the <u>S</u>elect... PushButton.</li> </ol>
<b>Graph Area</b> text in the Graph window	Select Node Fonts... from the Graph window Options PullDown menu.
<b>List Area</b> text in the Graph window	Select List Fonts... from the Graph window Options PullDown menu.
<b>Action Status Bar</b> in the Graph window	<ol style="list-style-type: none"> <li>1. Select Graph Window... from the Options PullDown menu to get the <b>Graph Window Settings</b> NoteBook.</li> <li>2. Select the Settings tab.</li> <li>3. Choose the <u>S</u>elect... PushButton.</li> </ol>

All of the above actions result in a **Font** dialog. You can select the font type, size, style and emphasis.

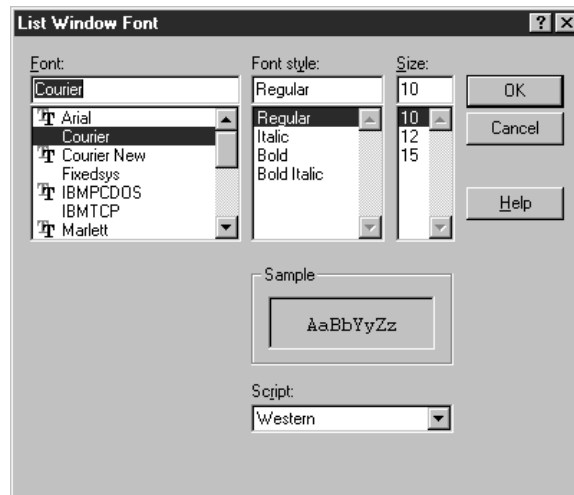


Figure 107. Graph Node Font Dialog

As you make selections, the **Sample Area** changes to preview the font definition you have chosen. Select the **OK** PushButton to accept the font changes. The new font settings will be saved to the Browser profile (ibrs.ini) when you close the Browser. Select the **Reset** PushButton to reset the dialog selections to the last used font definition.

## Browser: Load Database Facility

---

### Loading Files into the Browser



The **Load Database** dialog is used to load your program's information into the Browser. You can load the following types of files into a Browser session:

- .DLL Dynamic link library created using the linker option /BROWSE.
- .EXE Executable created using the linker option /BROWSE.
- .LIB Library file created using the **ilib** utility without the /NOBROWSE option.
- .PDB Browser database file created using the compiler option /Fb.
- .PDL Browser database file created from a loaded .LIB file.
- .PDE Browser database file created from a loaded .EXE file.
- .PDD Browser database file created from a loaded .DLL file.

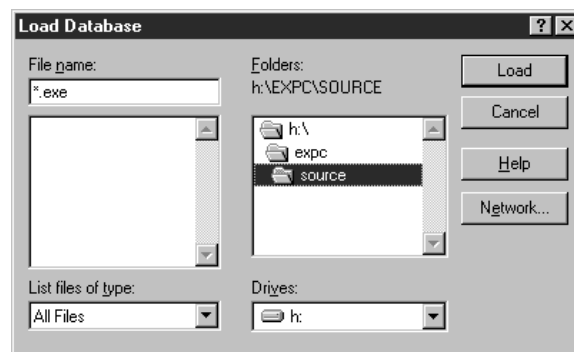


Figure 108. Load Database Dialog


For information on creating files to load into the Browser, see “Creating Files to Use with the Browser” on page 335.

To load a file:

1. Select Load... from the File PullDown menu to start the **Load Database** dialog.
2. Change the file name extension, if appropriate, in the **Open Filename:** TextEntry field.
3. Select the drive you want to load from using the **Drive:** DropDown list.
4. Select a directory on that drive from the **Directory:** ListBox.
5. Select the file name from the **File:** ListBox. Note that you can load more than one .PDB file at a time by making multiple .PDB selections in this ListBox. You cannot do a multiple load of any other file type.
6. Select the **Load** PushButton to load the information into the Browser.

**Note:** You can bypass these steps if you know the name and location of the file you wish to load. Enter the path name and file name into the **Open Filename:** TextEntry field.

## Browser: Merge Database Facility

You can also quickly load the classes that make up the IBM VisualAge for C++ Open Class Library by using the Load ► Cascade menu from the File PullDown menu. In addition, you can add your own files to the Load ► Cascade menu.  See “Adding Menu Items to the Load ► and Merge ► Cascade menus” on page 393.

When you select the **Load** PushButton, a **Progress** dialog is displayed to show you the amount of information that has been loaded by the Browser.

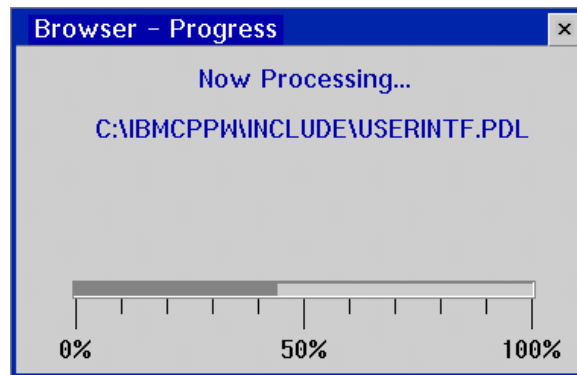


Figure 109. Progress Dialog

---

## Merging Files



The **Merge Database** dialog is used to load more than one program into a Browser session at a time. This is useful if you are thinking of adding classes from another program. You can check out the structure while still being able to look at your own program's structure.

When you browse a target program (an .EXE, .DLL, or .LIB), you will only see those classes, functions, and files that were actually used in the program. You will not see related objects. For example, assume that you have written a small program using the IBM User Interface class library, and it contains an **IFrameWindow**, a Menu bar, and some static text. The small program will only reference the classes, functions, and files of the **IFrameWindow**, **IMenuBar**, and **IStaticText** with their parent classes. If you want to add some PushButtons and a bitmap onto your window, you can see these classes by merging the User Interface Class Library data with your small program.

Also, many programs are written as an .EXE and one or more .DLLs. If you browse the .EXE, then you only see the data from that .EXE. You can merge in the data from the .DLL(s) and see the whole program's information.

## Browser: Merge Database Facility

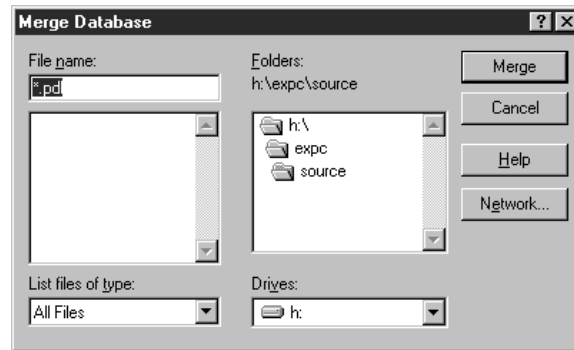



Figure 110. Merge Database Dialog

You can merge the following file types: .DLL, .EXE, .LIB, .PDB, .PDD, .PDE, and .PDL. If you merge more than one .PDB file, this is analogous to grouping a set of .OBJ files together into a single .LIB file, so the saved file version in this case is a .PDL file. For information on creating files to load into the Browser, see “Creating Files to Use with the Browser” on page 335.

To merge files:

1. Select Merge... from the File PullDown menu to start the **Merge Database** dialog.
2. Change the file name extension, if appropriate, in the **Open Filename:** TextEntry field.
3. Select the drive you want to load from using the **Drive:** DropDown list.
4. Select a directory on that drive from the **Directory:** ListBox.
5. Select the file name from the **File:** ListBox.
6. Select the **Merge** PushButton to merge the information into the current Browser session.

**Note:** You can bypass these steps if you know the name and location of the file you wish to merge. Enter the path name and file name into the **Open Filename:** TextEntry field.

You can also quickly load the classes that make up the IBM VisualAge for C++ Open Class Library by using the Merge ► Cascade menu from the File PullDown menu. In addition, you can add your own files to the Merge ► Cascade menu.  See “Adding Menu Items to the Load ► and Merge ► Cascade menus” on page 393.

When you select the **Merge** PushButton, a **Progress** dialog is displayed to show you the amount of information that has been merged into the current Browser database.



## Browser: Find Facility

**Note:** If you try to merge a file into the Browser database that duplicates some of the information that is already loaded into the Browser, a message will appear to inform you. This file will not be loaded into the Browser database.

---

### Finding Objects in the Current Window



Use the **Find** dialog to locate text in the current window starting from the currently selected object. To launch the **Find** dialog, select **Find...** from the Edit PullDown menu.



Figure 111. Find Dialog

Enter a text string into the **Find:** TextField or use the DownArrow icon to select from the last 10 text entries made.

Use the **Find** PushButton to initiate the search and close the **Find** dialog. Use the **Apply** PushButton to initiate the search and keep the **Find** dialog open. Use the **Cancel** PushButton to quit the dialog without performing the search.

If a match is found, the located text is brought into the view of the window and the text is highlighted. If no matches are found, a Message Box appears to let you know.

You can perform wildcard finds using:

- An asterisk (\*) to match any number of characters, and
- A question mark (?) to match one character.

You can use the **Case Sensitive** CheckBox to perform case dependent searches, and use the **Wrap Around** CheckBox to search the entire contents of the current window. The message “Wrapped” appears in the **Information Bar** when the find is starting to search from the top of the list.

## Browser: Search Facility

You can find the next instance of the text by either selecting Find Next from the Edit PullDown or use the **Ctrl-N** keys. Note that if you used the **Apply** PushButton, you can use it to find the next instance.

---

## Searching for Objects in the Entire Browser Database

You can use the **Search** dialog to search the proper names of objects. To launch the **Search Database** dialog, select Search... from the Actions PullDown menu.

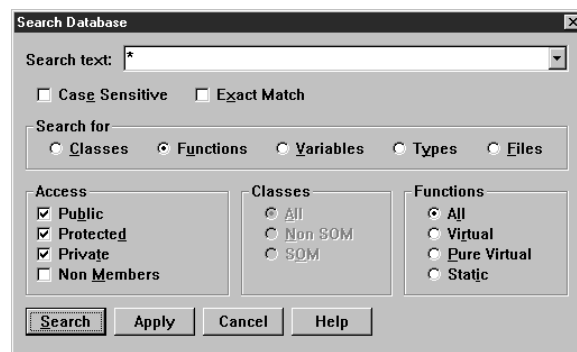


Figure 112. Search Database dialog

Enter the text string that you want to search for into the **Search** TextEntry field. You can also use the DownArrow icon to access the last 10 searches performed.

Select the **Search** PushButton to perform the search query and close the **Search Database** dialog. Select the **Apply** PushButton to perform the search query and keep the **Search Database** dialog available. Select **Cancel** to end the **Search Database** dialog without performing a query.

All program object names that match the search string will be listed in the List window. Note that the return types and arguments are not searched. If no matches are found, a Message Box appears to let you know.

You can perform wildcard searches using the following wildcards:

- A question mark (?) to signify specific character locations, and
- An asterisk (\*) to signify any number of character locations.

You can use the **Case Sensitive** CheckBox to perform case dependent searches. If you know the exact name of the object you want to locate, then use the **Exact Match** CheckBox. Note that you cannot use wildcards in conjunction with the **Exact Match** facility. The wildcard is treated as part of the actual search string.

## Browser: History Window

### An example using non-exact match:

- Enter foobar and deselect the **Exact Match** CheckBox.
- Results: foobar and realfoobar.

### An example using exact match:

- Enter foobar and select the **Exact Match** CheckBox.
- Results: foobar  
But not: realfoobar.

---

## The History Window



You can use the **History** window to redo previously invoked object-action pairs. This is useful if you replaced a particular List or Graph window contents and no longer have direct access to the object. To launch the **History** window, select History... from the Windows PullDown menu on any Browser window. For a definition of object-action pair, see page 331.

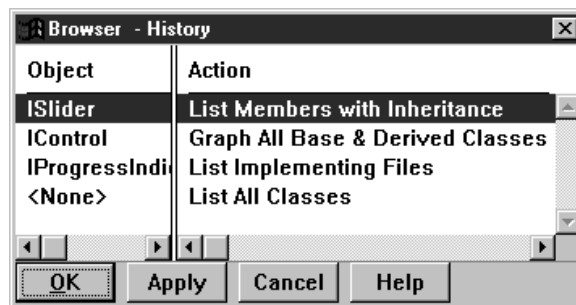


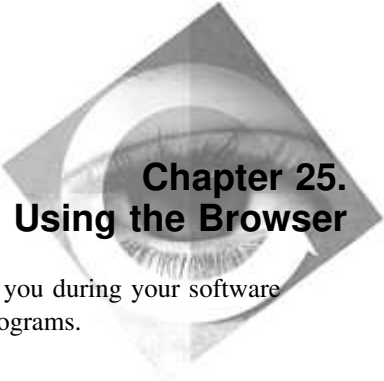
Figure 113. History dialog

The **History** window displays the last 40 unique object-action pairs that you have performed during your current session. The object is listed on the left hand side of the window and the action is displayed on the right side. Double-click on an object-action to invoke it, or select the **OK** PushButton to invoke the action and hide the window, or select **Apply** to invoke the action and keep the **History** window visible.

Invoking the object-action pair in the **History** window makes the Browser recalculate all the information. If the object-action pair that you want to invoke is listed at the top of the **History** window, use the Previous menu item from the Actions PullDown menu, or the **F6** key to initiate this command. The Browser does not have to recalculate the last object-action pair performed because the results are stored in a buffer. This method will be much faster.

## **Browser: History Window**

When you perform a load, merge, or refresh, the contents of the **History** window are checked to see that the object in each object-action pair is still valid. If it is not valid, it is removed from the **History** window list.



## Chapter 25. Using the Browser

This chapter describes how to use the Browser to help you during your software development cycle and to help you understand your programs.



The following are covered in this chapter:

- “Using the Browser to Assist in Development.”
- “Using the Browser to Aid Program Understanding” on page 378.
- “Using QuickBrowse” on page 389.

---


### Using the Browser to Assist in Development

The Browser can be a very useful tool during program development. It provides quick access to source files for:


 “Editing and Viewing Source Files.”


It gives you the ability to quickly view your uncompiled source files and the classes that make up the IBM VisualAge for C++ Open Class Library:

 “Browsing without Recompiling” on page 376.

 “Browsing the IBM VisualAge for C++ Open Class Library” on page 376.

It helps you in the design process by providing quick access to the VisualAge for C++ Open Class Library documentation and allows you to view more than one program at the same time.

 “Showing VisualAge for C++ Open Class Library Documentation” on page 377.

 “Browsing More Than One Program or Library at a Time” on page 377.

### Editing and Viewing Source Files

If you have ever worked on a large project, you know how difficult it is to keep track of where program elements are defined. Sometimes large programs can be split across several files and several directories. Even using traditional search methods, such as **grep**, it can take a long time to locate where particular program elements are defined.

The Browser can help you solve this problem in one simple step:

- Select the program element that you want to edit, and use the Object PopUp menu to select the Edit Definition action. This is the default setting, so you can also just double-click on the object to launch the edit session.


## Browser: Assisting in Development

The Browser will load the VisualAge for C++ Editor and take you into the program file at the exact location where the program element is defined.

You can also quickly view your source files without searching for a specific object definition in two ways:

- Use the List All Files or Show Include File Graph menu item from the Actions PullDown menu, or
- Select a class object and from its PopUp menu, select the List Implementing Files option to get a view of file objects.

Once you have a file object, use the file's PopUp menu to select the Edit File menu item in order to launch the VisualAge for C++ Editor with this file loaded.


 The various PullDown menus are described in “PullDown Menus” on page 416.

## Browsing without Recompiling

If you are working with code that does not compile, you can use the Browser to see the program elements and their relations without having a compiled executable file. In addition, if you make modifications to your program files, you can see these changes reflected in the Browser database without having to recompile. A utility called QuickBrowse quickly examines the makefile for your loaded project and derives the compile options used to compile the required source files. The source files are then quickly parsed as if they were compiled with such options. This method is much quicker than waiting for a lengthy compile, as well as useful when you inherit code from someone else, and it does not compile.

**Note:** The QuickBrowse feature is only available when the Browser is started from an IBM WorkFrame project.

You will not get as much information as if you had compiled and linked your source files first: no call, exception, or template instantiation information will be available. However, you can see the class structure of your program.

 For more information on QuickBrowse, see “Using QuickBrowse” on page 389.


## Browsing the IBM VisualAge for C++ Open Class Library

The Browser provides quick access to the classes that make up the IBM VisualAge for C++ Open Class Library. You can select them to view independently of any other files by selecting the library from the Load ► Cascade menu on the File PullDown menu.

The Browser also makes it easy to merge the classes that make up the IBM VisualAge for C++ Open Class Library when you are viewing your own programs.

## Browser: Assisting in Development


You can select the classes from the Merge ► Cascade menu on the File PullDown menu.

 The **Load Database** dialog is described in “Loading Files into the Browser” on page 368. The **Merge Database** dialog is described in “Merging Files” on page 369. The File PullDown menu items are defined in “File PullDown Menu” on page 417.

### Browsing More Than One Program or Library at a Time


When you browse a target program (an .EXE, .DLL, or .LIB), you will only see those classes, functions, and files that were actually used in the program. You will not see related objects. For example, assume that you have written a small program using the user interface classes defined in the IBM VisualAge for C++ Open Class Library, and it contains an **IFrameWindow**, a menu bar, and some static text. Next, you want to add a couple of PushButtons and a Bitmap onto your window. To see these classes, you can merge the User Interface Classes data (all of it) with your program's data, and see all the interface facts about these particular classes that make up part of the IBM VisualAge for C++ Open Class Library.

Also, many programs are written as an .EXE and one or more .DLLs. If you browse the .EXE, then you only see the data from that .EXE. You can merge in the data from the .DLL(s) and see the whole program's information.

Note that all IBM VisualAge for C++ Open Class Library classes start with the letter “I” with the exception of the I/O Stream and Complex Mathematics classes.  The File PullDown menu is described in “File PullDown Menu” on page 417. The **Merge Database** dialog is described in “Merging Files” on page 369.

### Showing VisualAge for C++ Open Class Library Documentation

The Browser gives you quick access to the class and function references provided for the IBM VisualAge for C++ Open Class Library. If you want to find out more information on one of these classes or functions, from the **Class** or **Function** PopUp menu, select the Show Documentation option.

 The PopUp menus are defined in “PopUp Menus” on page 424.












## Browser: Aiding in Program Understanding

---




### Using the Browser to Aid Program Understanding

The Browser provides two kinds of windows for displaying your program elements and their association with one another: List and Graph windows.

Many aspects of your program can be described by listing some group of elements in a List window:

-  “List All Classes Defined in the Currently Loaded Program.”
-  “List All Files Used to Create the Currently Loaded Program.”
-  “Listing All Objects Defined in a File” on page 379.
-  “Listing All Friends of a Class” on page 380.
-  “Listing All Friendships of a Class or Function” on page 381.
-  “Listing Immediate Callers and Callees for a Function” on page 382.
-  “Listing Instantiations of Classes or Functions” on page 385.
-  “Listing Implementing Files” on page 380.
-  “Listing All Class Members” on page 383.
-  “Listing Overriding Derived Classes” on page 384.
-  “Listing All the Exceptions That A Function May Encounter” on page 385.

Many aspects of your program can be described as relationships in a Graph window:

-  “Viewing Class Relationships” on page 386.
-  “Viewing Call Chains” on page 387.
-  “Viewing Include File Relationships” on page 388.

### List All Classes Defined in the Currently Loaded Program


By default, when you first load a program into the Browser, a list of all the classes defined for that program is displayed in a List window. If you no longer have this list visible, you can list all the classes by either:

- Selecting the List All Classes from the Actions PullDown menu, or
- Selecting the List All Classes from the List or Graph window **Background** PopUp menu.

### List All Files Used to Create the Currently Loaded Program

You can list all the files that were used to create the currently loaded program:

- Select the List All Files from the Actions PullDown menu
- Select the List All Files from the List or Graph window **Background** PopUp menu.

Note that the directories displayed in the list may not be accurate. These are the directories that were used when the program was created.  For more information on where the Browser searches for files, see “Changing Paths Used by the Browser” on page 363.



## Browser: Aiding in Program Understanding

### Listing All Objects Defined in a File

If you have a list of files or an include file graph, then you can use the file object to determine all the program elements that are defined in that file.

1. Select the file object with Mouse Button 2 to get the **File** PopUp menu.
2. Select the List Defined Objects menu item.

A list of all the program elements defined in the file will be displayed in a List window.

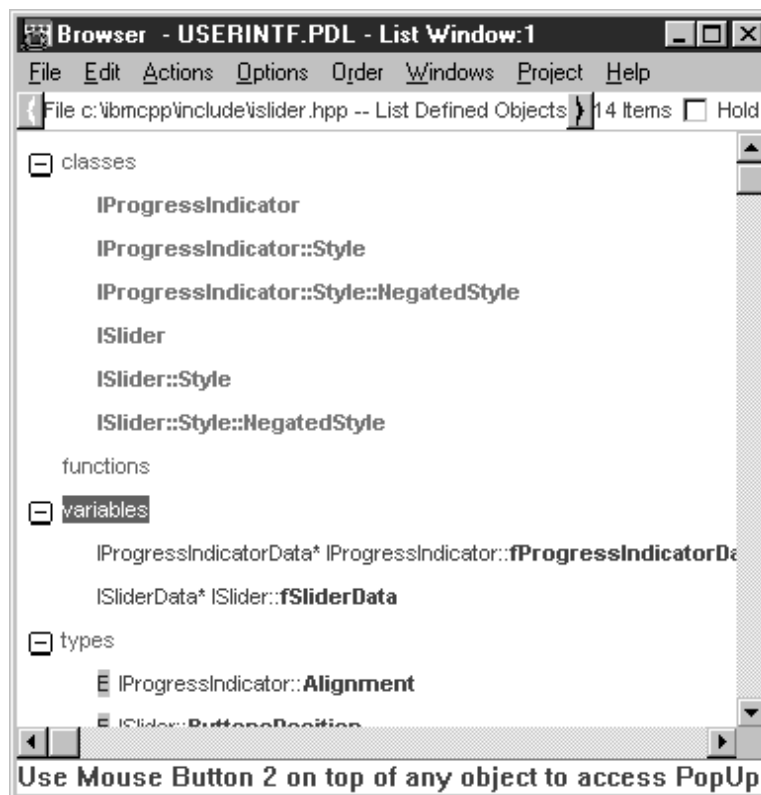


Figure 114. An Example List: File islider.hpp - List Defined Objects

## Browser: Aiding in Program Understanding

### Listing Implementing Files

If you want to know where class definitions are defined:

1. Select the class object with Mouse Button 2 to get the **Class PopUp** menu.
2. Select the List Implementing Files menu item.

A list of all files that contain definitions for the currently selected class and all of its members will be displayed in a List window.

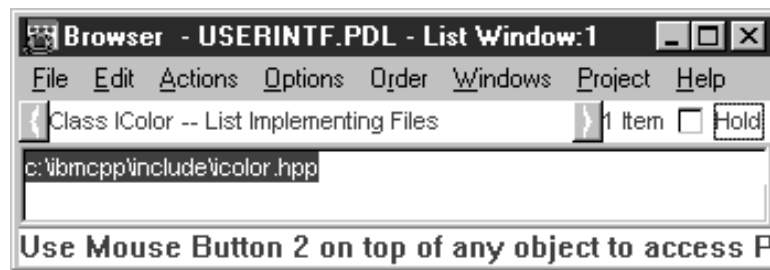


Figure 115. An Example List: Class IColor - List Implementing Files

### Listing All Friends of a Class

If you have a list of classes or an inheritance graph, you can use the class object to display all the friends of the currently selected class.

1. Select the class object with Mouse Button 2 to get the **Class PopUp** menu.
2. Select the List Friends menu item.

A list of all the friends defined for the currently selected class are displayed in a List window. If there are no friends defined for the currently selected class, then this menu item will be disabled.

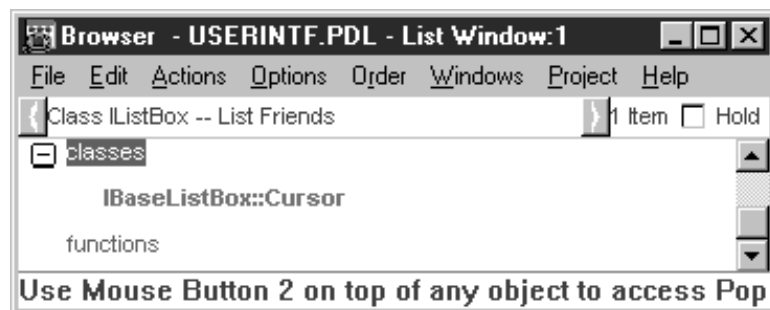


Figure 116. An Example List: Class IBaseListBox - List Friends

## Browser: Aiding in Program Understanding

### Listing All Friendships of a Class or Function

Friendships can be granted to either a single function or a member function, or to all the member functions of a class at once. You can list all the friendships defined for a class or function:

1. Select the class or function object with Mouse Button 2 to get either the **Class** or **Function** PopUp menu.
2. Select the List Friendships menu item.

A list of all the friendships that are defined for the selected object are displayed in a List window. If there are no friendships defined for the currently selected function, then this menu item will be disabled.

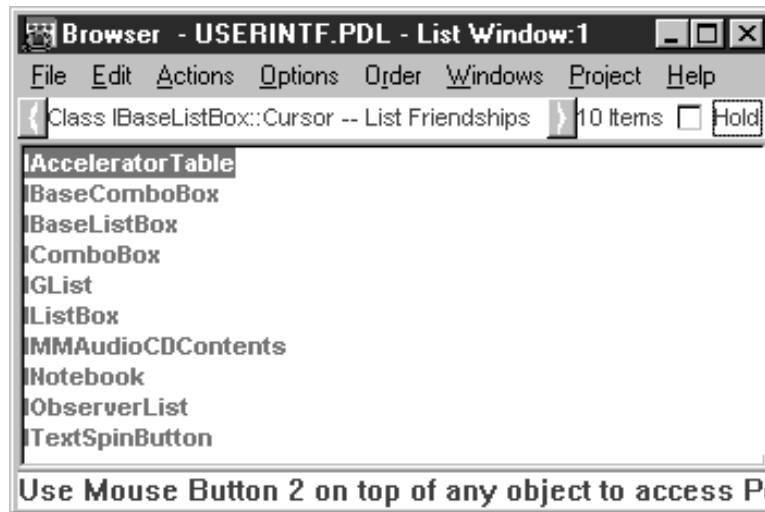


Figure 117. An Example List: Class IListBox::Cursor - List Friendships

## Browser: Aiding in Program Understanding

### Listing Immediate Callers and Callees for a Function

When debugging your programs, it is often important to know what impact changing a particular function may have on other functions that it calls or call it. You can list all the functions that call a particular function and that a particular function calls:

1. Select the function object with Mouse Button 2 to get the **Function** PopUp menu.
2. Select the List Immediate Callers & Callees menu item.

A list of all the callers and callees for the selected function will be displayed in a List window. If there are no callers or callees are defined for the currently selected function, then this menu item will be disabled.

**Note:** This information is not available if you have used QuickBrowse.

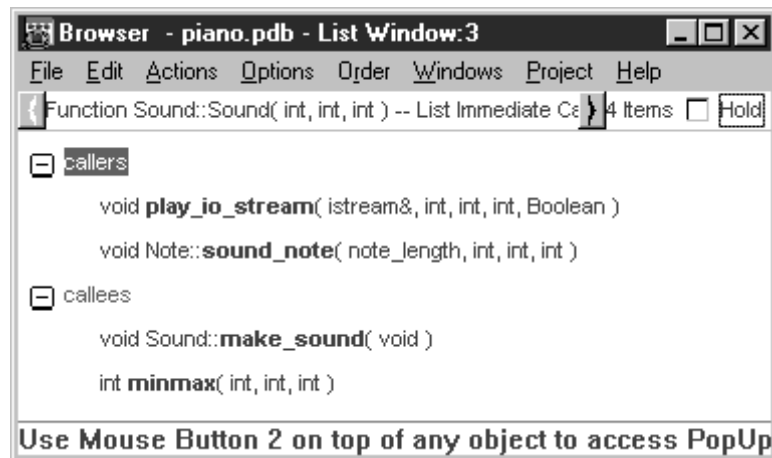


Figure 118. An Example List: Function Sound::Sound - List Immediate Callers & Callees

## Browser: Aiding in Program Understanding

### Listing All Class Members

It is often necessary to know what members are defined for a given class or what class a given member function is a member of. You can list all members of a class and its base classes:

1. Select the class or function object with Mouse Button 2 to get either the **Class** or **Function** PopUp menu.
2. Select the List Members with Inheritance or List Class Members with Inheritance menu item.

A List window container view of all the classes, base classes, and members is displayed. A container view is a list which can be further expanded using the + and - icons to expand and collapse the entries. There are three ways to order this window: by classes, by access, or by type. By default, the items are ordered by class. Note that this list is not ordered alphabetically, but are arranged in a depth first tree traversal of the classes' inheritance hierarchy.

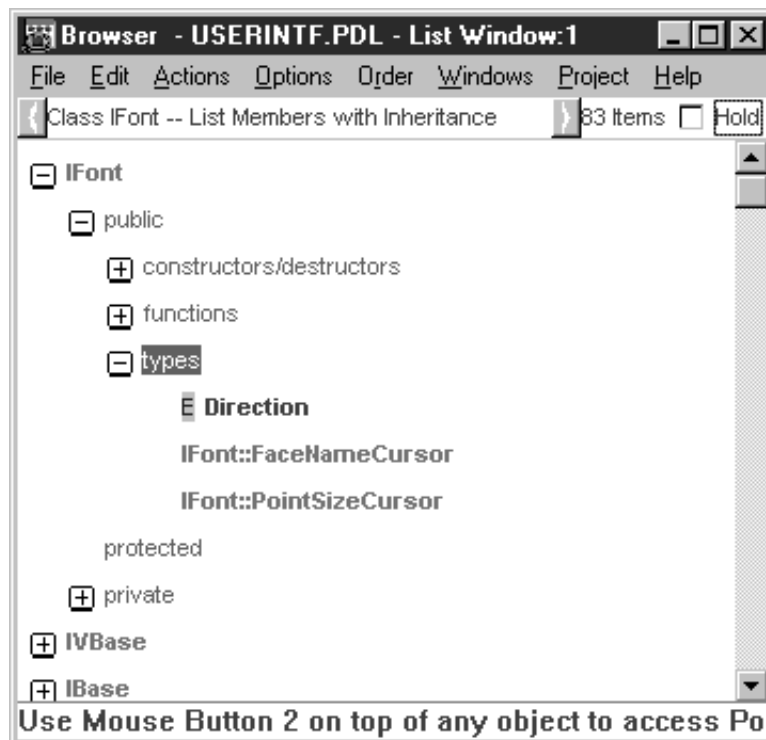


Figure 119. An Example List: Class IFont - List Members with Inheritance

## Browser: Aiding in Program Understanding

### Listing Overriding Derived Classes

It is often important to know when, where, and if a function is overridden. You can list all the overriding derived classes for a function:

1. Select the function object with Mouse Button 2 to get the **Function PopUp** menu.
2. Select the List Overriding Derived Classes menu item.

A list of all the overriding derived classes will be displayed in a List window. This menu item is disabled if the function is not the member of a class, or there are no derived classes for the class this function is a member of.

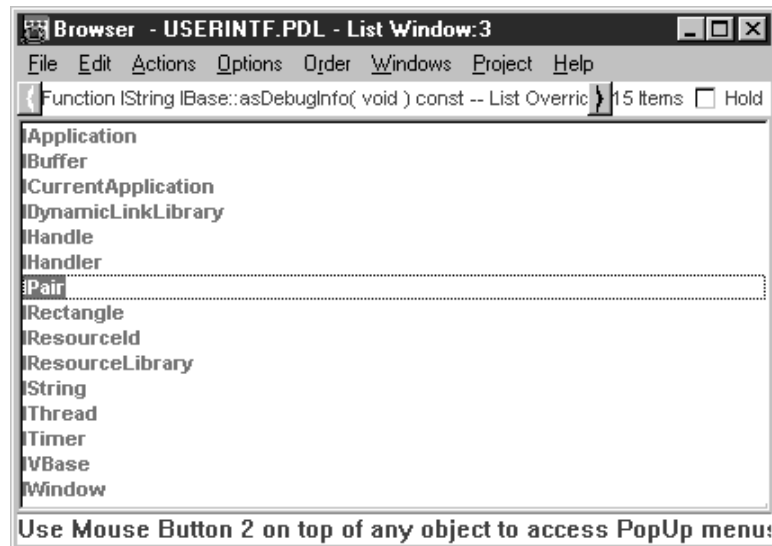


Figure 120. An Example List: Function IBase::asDebugInfo - List Overriding Derived Classes

## Browser: Aiding in Program Understanding

### Listing Instantiations of Classes or Functions

You can list the template instantiations for classes and functions:

1. Select the class or function object with Mouse Button 2 to get either the **Class** or **Function** PopUp menu.
2. Select the List Instantiations menu item.

A list of all the template instantiations for the currently selected object will be displayed in a List window. This menu item is disabled if the currently selected class or function is not a class or function template.




Figure 121. An Example List: Class ISet<class Element> - List Instantiations

### Listing All the Exceptions That A Function May Encounter

C++ uses exception handling to support error handling because throwing or catching an exception can affect the way a function relates to other functions. You need to know what these exceptions are.

You can quickly list all the exceptions for a given function using the List Possible Exceptions Thrown item on the **Function** PopUp menu.

**Note:** This information will not be available for any data loaded using QuickBrowse.

 The **Function** PopUp menu is described in “Object PopUp Menu Items” on page 425.

## Browser: Aiding in Program Understanding

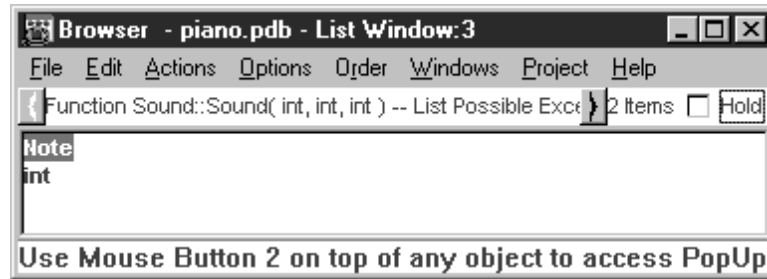



Figure 122. An Example List: Function Sound::Sound - List Possible Exceptions Thrown

## Viewing Class Relationships

You can quickly graph the class inheritance relationships using one of the following Class PopUp menu items:

- Graph All Base & Derived Classes
- Graph All Base Classes
- Graph All Derived Classes
- Graph Immediate Derived Classes

 The Class PopUp menu is described in “Object PopUp Menu Items” on page 425.

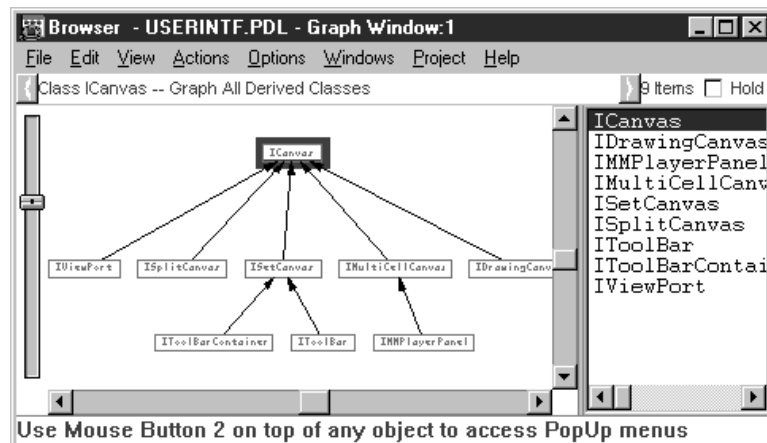


Figure 123. An Example Graph: Class ICanvas - Graph All Derived Classes




## Browser: Aiding in Program Understanding

### Viewing Call Chains


When debugging your programs, you often have to follow the call chain. Sometimes you may have a function that is acting unexpectedly. You need to know all the functions that may be calling it or that it calls, so that you can determine what effect the function may have on other program components.

You can quickly graph the call chain of a function using one of the following **Function** PopUp menu items:

- Graph All Callers
- Graph All Callees
- Graph All Callers & Callees
- Graph Immediate Callers & Callees

depending on how much of the call chain you want to see. You will not see calls to C functions defined in the system header files.  See “/Fb” on page 123 for a description of how to compile for use with the Browser.

**Note:** This information will not be available for any data loaded using QuickBrowse.

 The **Function** PopUp menu is described in “Object PopUp Menu Items” on page 425.

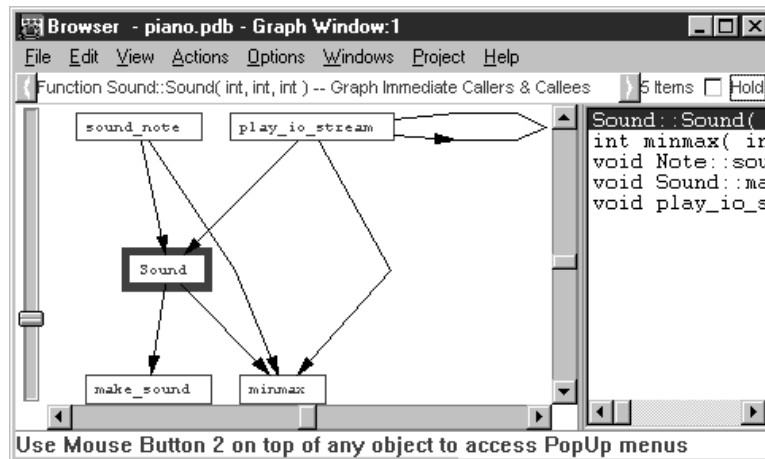



Figure 124. An Example Graph: Function Sound::Sound - Graph Immediate Callers & Callees

## Browser: Aiding in Program Understanding

### Viewing Include File Relationships

You can quickly graph the file structure of your programs showing you where header files are included. You can show this structure using the following PopUp menu items:

- Graph All Includees
- Graph All Includers
- Graph All Includers & Includees

 The **File** PopUp menu is described in “Object PopUp Menu Items” on page 425.

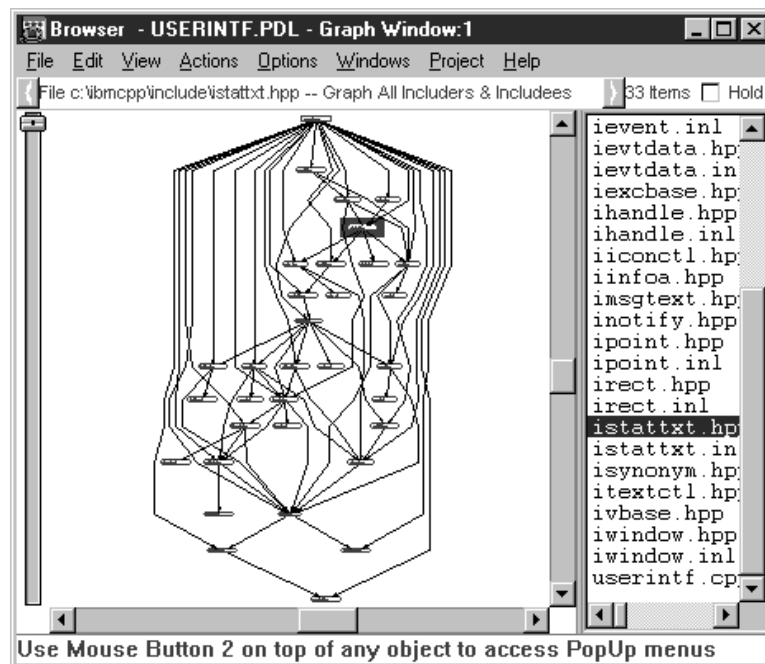



Figure 125. An Example Graph: File istat.txt.hpp - Graph All Includers & Includees

When the Browser lists files, it displays the path name of the files when the program was compiled. However, this path name may not be correct, as is the case with the Browser shipped .PDL files for the IBM VisualAge for C++ Open Class Library classes that you can load or merge from the Load ► and Merge ► Cascade menus.

 For information on where the Browser searches for files, See “Changing Paths Used by the Browser” on page 363.

---

## Using QuickBrowse



QuickBrowse is a code analysis technology that allows the VisualAge for C++ Browser to extract declarations from C++ source code without the associated overhead of compilation. Declarations local to functions and function call information are not provided by QuickBrowse.

The QuickBrowse feature allows you to quickly obtain and browse declarations for code for which there is no compiler generated (**/Fb**) Browser information. Use QuickBrowse for the following reasons:

- It is faster than compiling the code
- You may be able to browse files that do not compile

**Note:** The QuickBrowse feature is only available when the Browser is started from an IBM WorkFrame project.

QuickBrowse parses the top level declarations which must be valid C++ statements, and ignores the bodies of function definitions.


You may want to use QuickBrowse if you are not interested in function call information. Also, if you have code where the declarations is well defined, but function bodies will not compile, you can browse the type information with QuickBrowse.

If you are browsing in a project, and the Browser detects that some, or all, information is missing, a dialog will appear telling you that this information is missing, and will give you the option of QuickBrowsing the files for which data is missing. Messages will appear in the Project's monitor, just as if you were doing a build.

Note that the QuickBrowse feature is not a complete replacement to the Generate Browser information (**/Fb**) compiler option. The speed of QuickBrowse does come at a cost in the richness of information provided. Since the QuickBrowse feature does not look inside of function bodies, function call, exception, and template instantiation information are not available. If you need to know this kind of information, then you will need to compile the file and use the Generate Browser information (**/Fb**) option.

## Browser: QuickBrowse

### What Do You See When QuickBrowse Starts

When QuickBrowse is started, you will be presented with the **Browser Files** dialog. It lists all the files that will require QuickBrowsing. In addition, you can select the **Change Path** PushButton to change the paths used by the Browser.  For more information on changing paths, see “Changing Paths Used by the Browser” on page 363. Make sure the **QuickBrowse files which could not be loaded** CheckBox is selected in order to QuickBrowse the files.

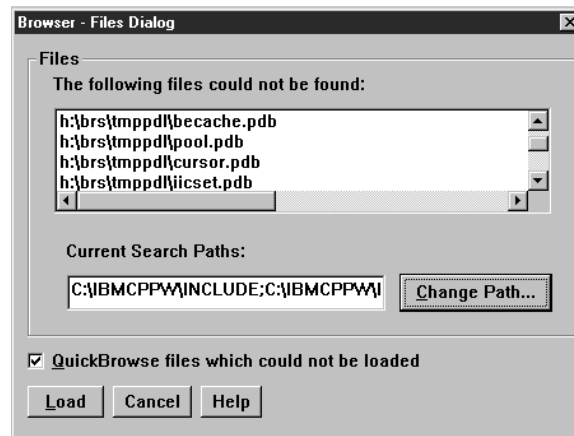



Figure 126. Browser Files Dialog

Select the **Load** PushButton on the **Browser Files** dialog to QuickBrowse the listed files. If the Browser cannot find the compiler option information for some of the files listed in the **Browser Files** dialog, then the **Browser QuickBrowse** dialog will appear.  For more information on where the Browser searches for files, see “Changing Paths Used by the Browser” on page 363.

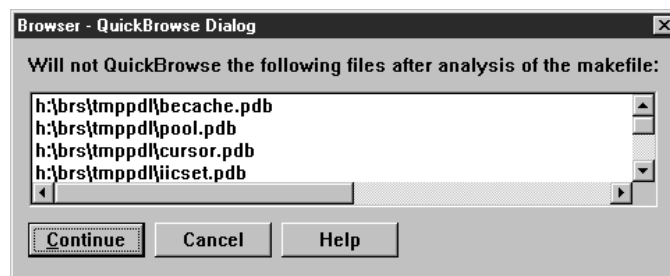


Figure 127. QuickBrowse Dialog

## Browser: QuickBrowse

The **QuickBrowse** dialog lists the files for which compiler information could not be found, but which require QuickBrowsing or recompiling. Select **Continue** to load the already QuickBrowsed information of the other files listed in the **Browser Files** without the information of these listed files.

### Scenarios for Using QuickBrowse

The following are some sample QuickBrowse scenarios:

- “Porting Code Scenario.”
- “Design Scenario.”
- “Browsing Libraries Scenario.”
- “Code Understanding Scenario.”

#### Porting Code Scenario

You may be porting from one operating system to Windows. Your types are correct, but many of your functions cannot compile, since they use system functions which do not exist on Windows. You can use the QuickBrowse feature on such code to understand the type structure.

#### Design Scenario

The QuickBrowse feature can help you with your designing, especially if you are doing design in a group, or you wish to communicate your design with others. To do this, you need to get your types straight - or at least to the point where they are “correct” C++ code. Then, you can use QuickBrowse.

You could compile at this point too, but QuickBrowse will generally be quicker, and the function bodies can be in whatever state you like, as long as the { } match up.

#### Browsing Libraries Scenario

Coming to terms with class libraries and frameworks means understanding the types they provide. Typically, intra-library function calls are not exposed to the user, except for trivial inline functions. You can create a single source file which includes all the interfaces to a third party library, and either use the compiler to generate the Browser information, or use the QuickBrowse feature.

#### Code Understanding Scenario

When you get new code, use QuickBrowse to understand the type structure of the code while ignoring the functional details. Then, when you want to look at the function call relationships, compile the code. In well-designed and well-built code, the key abstractions will be found in the type structure. Dealing with just the types also reduces the amount of information that you have to comprehend at the beginning.


## Browser: Updating the Browser Database

---

### Updating the Browser Database

While you are browsing your program, you may also be modifying the source files. By selecting the Refresh action from the File PullDown menu, the Browser will check to see if the loaded program is out of date by checking the dates of the source files against the program files. If you have modified your source code, but have not regenerated the Browser information, the **Browser Files** dialog appears.

- If you are browsing an IBM WorkFrame project:

You can choose to QuickBrowse your source by selecting the **Load** PushButton. The QuickBrowse facility is fast, but there may be a loss of information provided for your program. The benefit is that you do not have to wait for a lengthy recompile of your source to browse the declarations in your programs.  For more information on QuickBrowse, see “Browsing without Recompiling” on page 376 and “Using QuickBrowse” on page 389.

- If you are not browsing an IBM WorkFrame project:

The **Browser Files** dialog will list all files that are out of date and cannot be found. You cannot use the QuickBrowse facility to load these files, since QuickBrowse is only available for IBM WorkFrame projects. If you select the **Load** PushButton, the Browser will update the Browser database with the information that is available and will delete all the information associated with the listed files. For example, if you are browsing an .EXE file that has five .PDB files (call them A, B, C, D and E), and you have updated A and B, but not C and D, and E did not change, then the Browser will delete the old information (A, B, C and D), and load in the new (A and B). The result is that after the refresh, the Browser has information for A, B and E, but not for C and D. To regain the information for C and D, you will have to rebuild the .EXE file and load it into the Browser. If you do not want to loose the information contained in the files listed in the **Browser Files** dialog, then select the **Cancel** PushButton, and rebuild these listed files. Now you can select the Refresh action again to update all the information in the Browser database.

**Note:** The Refresh action will only load or modify the current Browser database as required by performing an incremental smart load. This means that the current Browser database will be refreshed much quicker than if it had to regenerate the entire Browser database.

---

### Adding Menu Items to the Load ► and Merge ► Cascade menus

If you perform repeated loads or merges of a particular program, you can add it to the Load ► and Merge ► Cascade menus for quick access. To do this:

1. Create an ASCII file called `brsmenu.txt` and place it in a directory in your DPATH.
2. Use the following format:


```
Menu Item Name"path_name\file_name
```

Where Menu Item Name is the name you want to have appear on the Load ► and Merge ► Cascade menus, and `path_name\file_name` is the path name and file name of the file to be loaded or merged. You can have spaces in the Menu Item Name. Be sure to separate the Menu Item Name from the `path_name\file_name` with a double quote (").

**Note:** You must have a blank line at the end of the `brsmenu.txt` file.

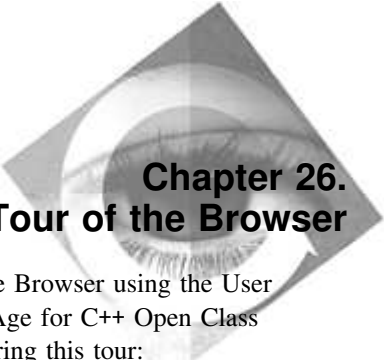
3. You can add upto a maximum of six files.

The new menu items will be added to the Load ► and Merge ► Cascade menus the next time you start the Browser.

 For more information on loading, see “Loading Files into the Browser” on page 368. For more information on merging, see “Merging Files” on page 369.

## **Browser: Adding Menu Items**





## Chapter 26. A Tour of the Browser



This tour takes you through some of the features of the Browser using the User Interface classes that make up part of the IBM VisualAge for C++ Open Class Library. The following is a list of tasks performed during this tour:

- “Starting the Browser and Loading User Interface Classes” on page 396
- “Finding A Class” on page 397
- “Showing the Inheritance Relationship of a Class” on page 398
- “Finding Another Class” on page 399
- “Changing the View of a Graph” on page 400
- “Investigating the Members of a Class” on page 401
- “Customizing Program Elements” on page 402
- “Editing Files from the Browser” on page 402
- “Organizing the Information in a List Window” on page 404
- “Finding A Function” on page 405
- “Showing the VisualAge for C++ Documentation for a Particular Function” on page 406
- “More About the PopUp Menu Actions” on page 406
- “Invoking Actions Again” on page 407
- “Graphing Include File Relationships” on page 407
- “Returning to Previous Queries/Displays” on page 408
- “Keeping Your Windows From Being Replaced” on page 409
- “Changing the Default Settings for List and Graph Windows” on page 410
- “Manipulating Graphs” on page 410
- “The Browser and WorkFrame” on page 411

## Browser: A Tour

---

### Starting the Browser and Loading User Interface Classes


You will be browsing the User Interface classes of the IBM VisualAge for C++ Open Class Library.

1. On the command line, enter: `ibrs`.
2. Select the File PullDown menu.
3. Select the Load ► Cascade menu.
4. Select the User Interface Classes menu item.

By default, the List Window will populate with a list of all the classes defined in the User Interface classes of the IBM VisualAge for C++ Open Class Library.



Figure 128. List window showing all classes in the User Interface classes

 For more information on loading, see “Loading Files into the Browser” on page 368.

## Finding A Class

Suppose that in your application, you want to use a Listbox. You will need to find more about Listboxes. From the List window:

1. Select the Edit PullDown menu.
2. Select the Find... menu item. The **Find** dialog appears.
3. Enter `listbox` as the search text.
4. Check that case sensitive is off.
5. Select the **Apply** PushButton until `IListBox` is highlighted.
6. Cancel the **Find** dialog.

The list scrolls to the `IListBox` class.

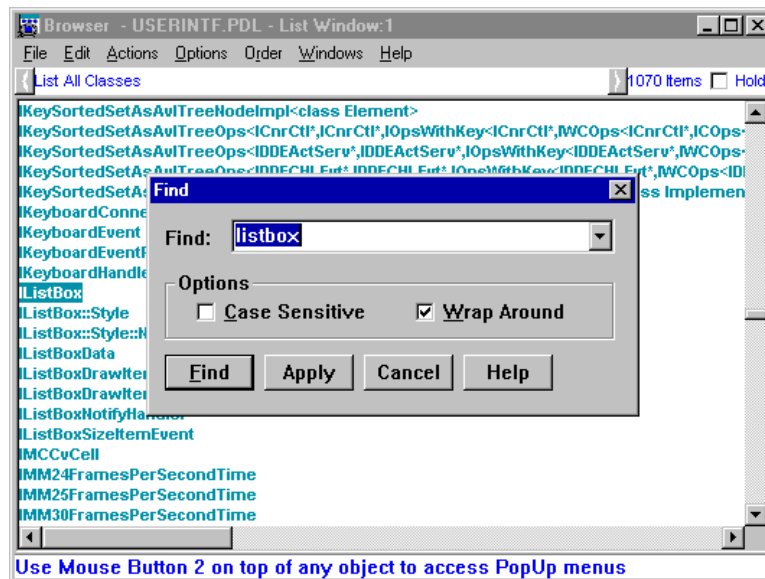



Figure 129. Finding a class name

 For more information on the **Find** facility, see “Finding Objects in the Current Window” on page 371.

## Browser: A Tour

### Showing the Inheritance Relationship of a Class

Now you will need to show the relationship of IListBox in the class hierarchy. Note that there are other classes that start with IListBox which you could look at later.

1. Click Mouse Button 2 on top of IListBox to get the **Class** PopUp menu.
2. Select Graph All Base Classes from the **Class** PopUp menu.

The Graph window shows the inheritance hierarchy for IListBox, and lists an alphabetical list of the classes that appear in the graph.

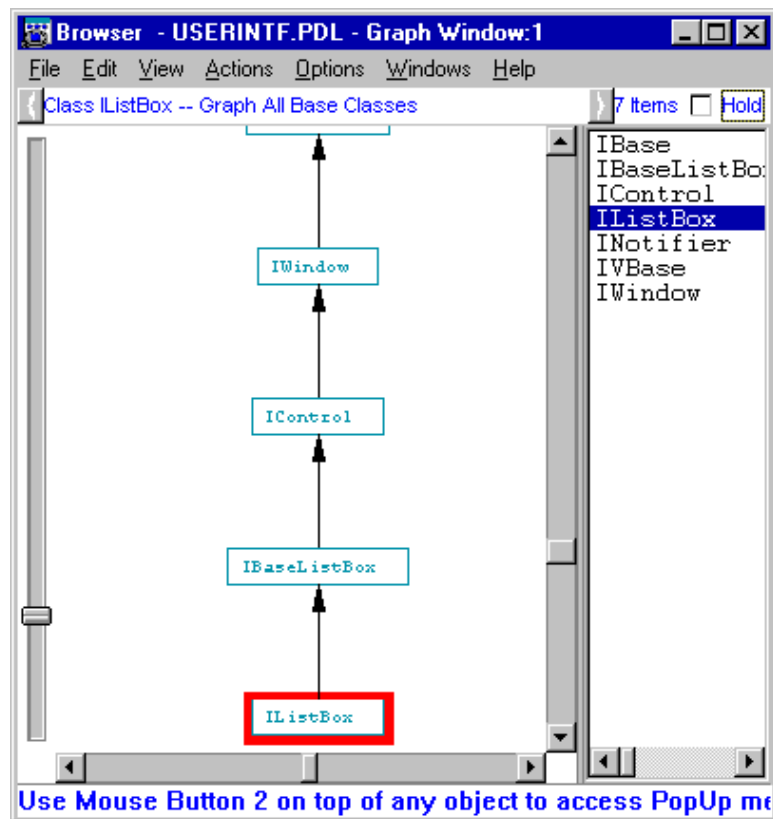


Figure 130. Graph window showing all base classes of the IListBox class

## Finding Another Class

Now you may want to investigate what other controls are in the User Interface classes of the IBM VisualAge for C++ Open Class Library. Maybe there is a special color Listbox.

1. Select the IControl class object (either using the node on the graph or the alphabetical graph-list item) using Mouse Button 2. The **Class PopUp** menu appears.
2. Select the Graph All Base & Derived Classes menu item.

The new graph shows all the classes that inherit from IControl. However, the graph is too large to view all at once.

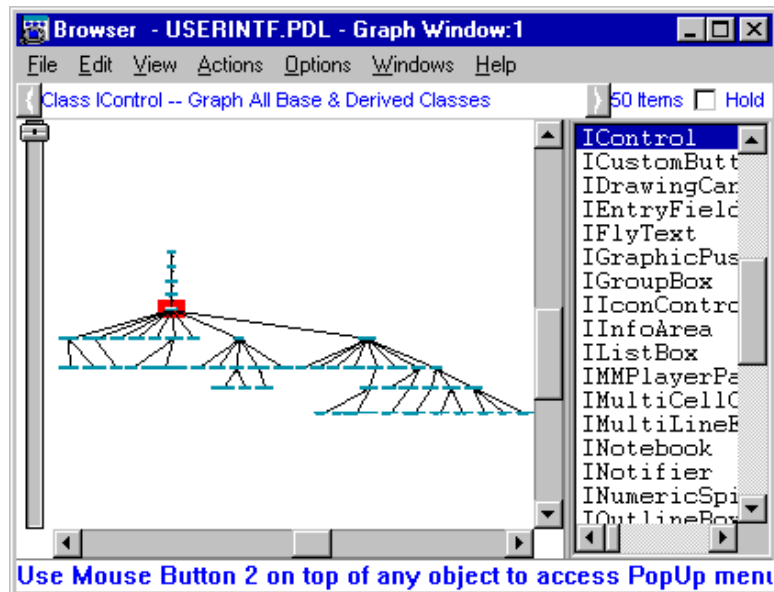


Figure 131. Graph window showing all base and derived classes of IControl class

## Browser: A Tour

### Changing the View of a Graph

Some graphs are better viewed using a horizontal organization rather than vertical (the default). For example, wide trees with long names are better shown horizontal, while tall trees with short names are better shown vertical. The inheritance graph of `IControl` falls into this category. To change the organization of the nodes from vertical to horizontal:

1. Click Mouse Button 2 on the background of the Graph window to display the **Background** PopUp menu. (Note that the List window also has a **Background** PopUp menu).
2. Select the Horizontal menu item.

Now adjust the zoom factor of the graph:

1. Display the Graph window **Background** PopUp menu again.
2. Select the Max Zoom in menu item.
3. Display the PopUp menu again
4. Select the Center menu item in order to center the selected node (Class `IControl`) in the **Graph Area** of the Graph window.

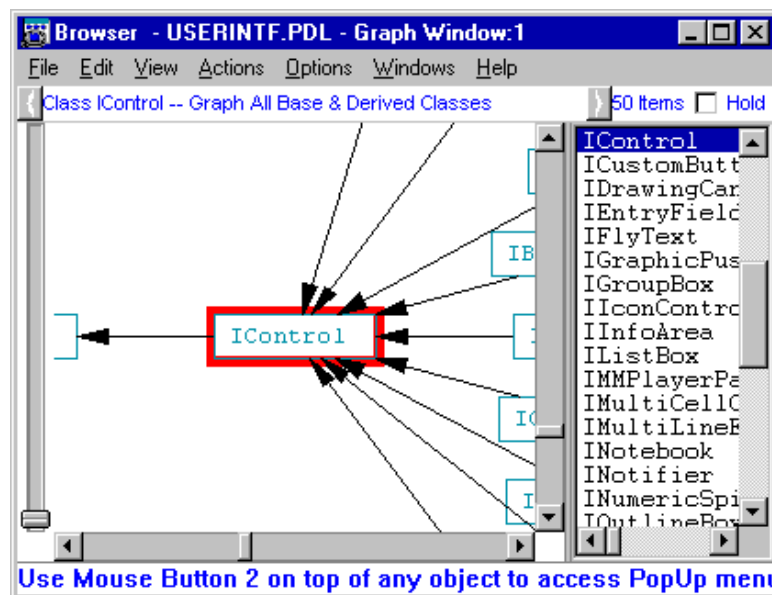


Figure 132. Graph window showing all base and derived classes of `IControl` class

Note the difference between the vertical organization of the nodes in the previous graph and the horizontal organization in this graph.

## Investigating the Members of a Class

Now you need to find out more details about `IListBox` and its members.

1. Select either the graph node or the graph-list item for `IListBox`.
2. Select List Members with Inheritance on the **Class** PopUp menu.

A List window appears showing the contents view of `IListBox`. This is called a container view. It can be expanded to show all the members of `IListBox` or any of the members of the base classes of `IListBox`.

1. Select the + icon on `IListBox` to expand and show the classes public, protected, and private sections.
2. Expand the + icon on public to show the public constructors, destructors, functions, variable, and types.
3. Expand the + icon on types.

Note how different colors and highlighting techniques are used for the various kinds of program elements.

Note that the classes are arranged in a depth first tree traversal of the classes inheritance hierarchy.

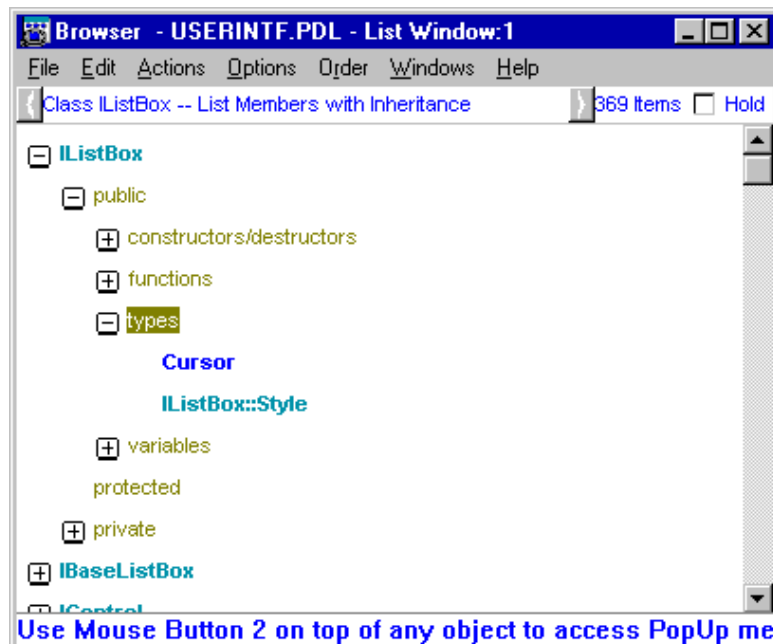



Figure 133. List window showing all members with inheritance of `IListBox` class

## Browser: A Tour

---

### Customizing Program Elements

Each program element has its own unique customization for either the Graph or List windows. You can associate different colors and shapes to different program element types. Also, each object has a double-click action associated with it, as indicated by a check mark on the PopUp menu.  For information on changing the Graph or List window settings, see “Changing the Default Graph Window Settings” on page 356 or “Changing the Default List Window Settings” on page 345.

In the List window, some listed items have a one-letter attribute, and bold is used to highlight function names from the rest of the string. The one letter attributes are: V-Virtual (on functions), PV-Pure Virtual (on functions), C-Const (on functions), S-Static (on functions and variables), and E-Enum (on types).

---

### Editing Files from the Browser

The default double-click action for class and function objects is to edit the file containing the object's definition. You can make changes to the object's definition or just read the code/comments in the source code.

1. Double-click on `IListBox` to load `ilistbox.hpp` into the VisualAge for C++ Editor. Note how the editor places you at the start of the class definition.
2. Close the editor.



## Browser: A Tour

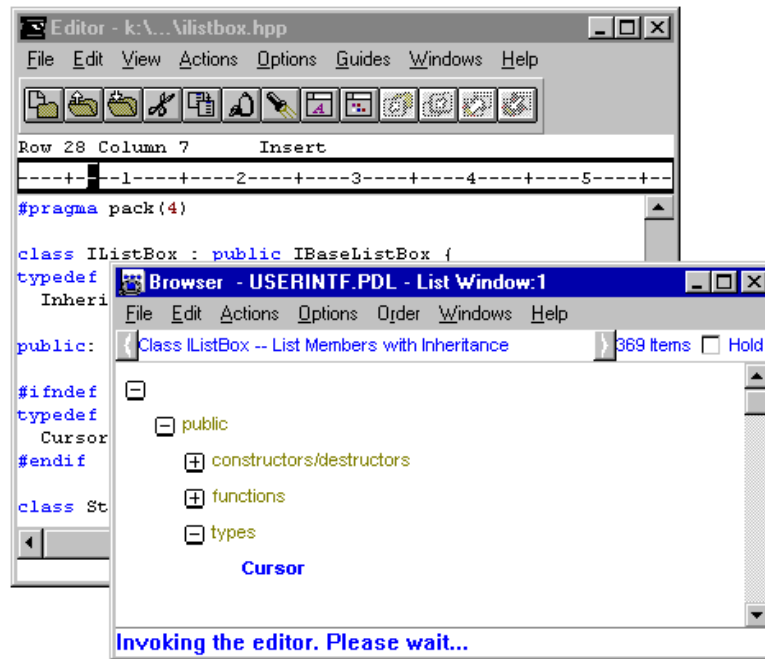


Figure 134. List window and VisualAge for C++ Editor

## Browser: A Tour

### Organizing the Information in a List Window


You can organize the information about `IListBox` in different ways. For example, suppose that you are only interested in the public members:

1. Select the Order PullDown menu.
2. Select the Access menu item to change the order of the list items. Previously, the order was by Class; the class was at the highest level, followed by the access type (public, protected, private), followed by the program element type. Now, the access methods are placed at the highest level, followed by program element type, followed by class.
3. Double click on public to expand the whole tree under that label.
4. Display the List window PopUp menu by clicking Mouse Button 2 on the List window background.
5. Select the Expand All menu item.



Figure 135. List window showing organization by access

Compare the order of the contents in this List window with the previous List window contents.

 For more information on ordering a container view, see “Ordering the Contents of a Container View” on page 342.

## Finding A Function

Now search for all the functions that contain the word `color`.

1. Select the Actions PullDown menu.
2. Select the Search... menu item. The **Search Database** dialog appears.
3. Type `color` as the search text.
4. Select **Functions** as the objects to search for.
5. Select **Public** as the access type.
6. Select **All** as the type of function.
7. Select the **Search** PushButton.

A list of all the function names that have the text string `color` are displayed. Scroll the list of functions until you find a function that allows you to set the color (`setBackgroundColor`).

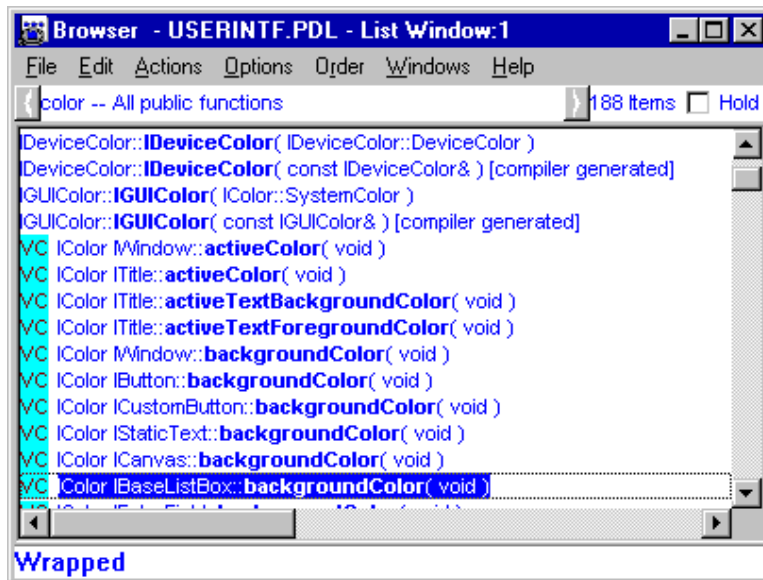



Figure 136. List window showing all public functions that contain the “color” string

 For more information on the **Search Database** facility, see “Searching for Objects in the Entire Browser Database” on page 372.

## Browser: A Tour

---

### Showing the VisualAge for C++ Documentation for a Particular Function

To find out more information about the `setBackgroundColor` function with regards to Listboxes:

1. Click Mouse Button 2 on the `IListBox::setBackgroundColor` function.
2. Select the Show Documentation menu item from the **Function** PopUp menu.  
The VisualAge for C++ online reference manual opens to this function.

---

### More About the PopUp Menu Actions


You can also perform other actions from the various PopUp menus.

On the **Function** PopUp menu, select any of:

- Graph All Callers & Callees to display the call graph for that function. For an example, see “Viewing Call Chains” on page 387.
- List Possible Exceptions Thrown to see the exceptions that could be thrown by this function. For an example, see “Listing All the Exceptions That A Function May Encounter” on page 385.
- List Overriding Derived Classes to see the derived classes that override this function. For an example, see “Listing Overriding Derived Classes” on page 384.

From the **Function** or **Class** PopUp menus, select:

- List Friends, List Friendships, or List Instantiations of a template. Note you can only list friends from a class, not a function. For examples, see “Listing All Friends of a Class” on page 380, “Listing All Friendships of a Class or Function” on page 381, and “Listing Instantiations of Classes or Functions” on page 385.

The above actions help you to understand a class library, someone else's code, or your own programs.  For more information, see “PopUp Menus” on page 424.

---

## Invoking Actions Again

The last action done in any window is always saved. For example:

1. Select the Actions PullDown menu.
2. Select the List All Files menu item. A list of all the files used to create the User Interface classes of the IBM VisualAge for C++ Open Class Library are listed in a List window.
3. Select the Actions PullDown menu again.
4. Select the Previous menu item. The results of your last action are displayed.
5. Select the **F6** key to go back to the list of files.

By selecting the Previous or the **F6** key, you can toggle back and forth between displays.

---

## Graphing Include File Relationships

To find out where classes, types, variables, and code included in your program come from, you will need to look at an include file relationship.

1. Click Mouse Button 2 on the `igbitmap.hpp` file to invoke the **File** PopUp menu.
2. Select the Graph All Includers & Includees menu item.

A graph of the include file structure is displayed. This graph indicates which files include other files.

## Browser: A Tour

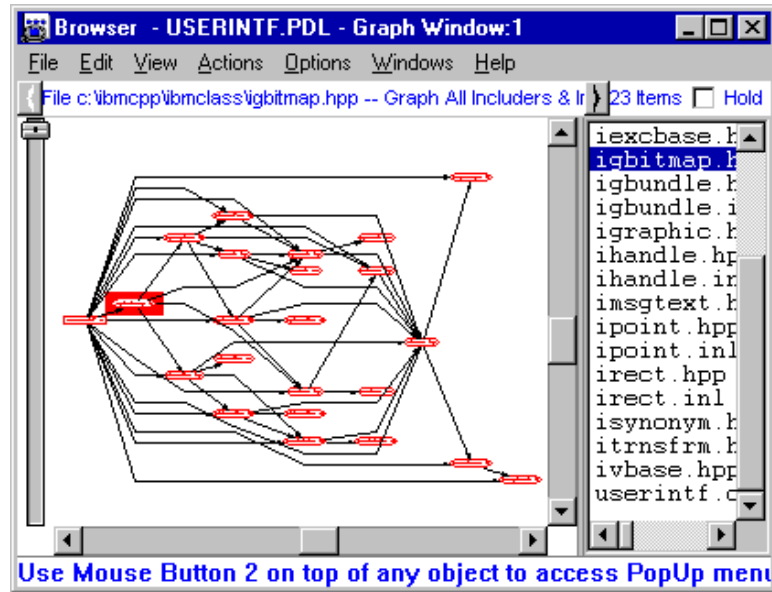


Figure 137. Graph window showing all includers and includees of igbitmap.hpp


---

## Returning to Previous Queries/Displays

If you performed a query or had a display that you want to return to, use the **History** window.

1. Select the Windows PullDown menu.
2. Select the History... menu item.

The last 40 object-actions that were performed are listed, and you can perform any of those actions again by double-clicking on the object-action pair.

 For more information on the **History** facility, see "The History Window" on page 373.

## Keeping Your Windows From Being Replaced

By default, the Browser replaces the contents of the current window. You can keep the current window and do the next action in a new window by using the **Hold** CheckBox. From the List window:

1. Select the Actions PullDown menu.
2. Choose the List All Files menu item.
3. Select the **Hold** CheckBox on this window.
4. Click Mouse Button 2 on `ilistbox.hpp` from the list of files to invoke the **File** PopUp menu.
5. Choose the List Defined Objects menu item. A new List window will open with the results. You can have a maximum of four List windows and four Graph windows open at any time.
6. Double-click on the classes label to see all the classes defined in `ilistbox.hpp`.

Use the Windows PullDown menu to get back to another window.

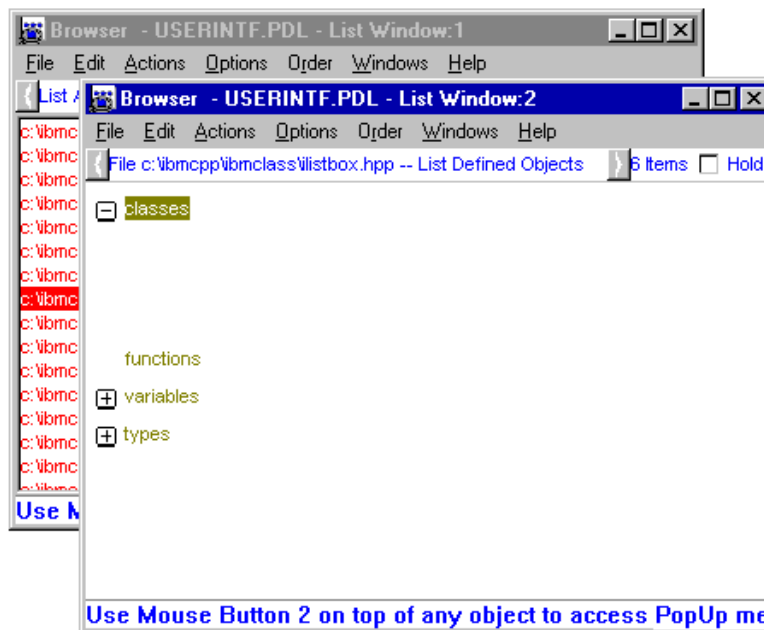






Figure 138. Two List windows open at same time

## Browser: A Tour

---

### Changing the Default Settings for List and Graph Windows

You can alter the default settings to the List and Graph windows by selecting:

- Fonts... to change the font of the List window.  See “Changing Fonts” on page 366.
- List Fonts... and Node Fonts... to change the fonts in the Graph window.  See “Changing Fonts” on page 366.
- List Window... to change the double-click actions for the objects in the List window, the colors used by the List window, and the text style used by the List window.  See “Changing the Default List Window Settings” on page 345.
- Graph Window... to change the double-click actions for the objects in the Graph window, the colors used by the Graph window, the node and line shape used by the graph, and the size of the bitmap to save.  See “Changing the Default Graph Window Settings” on page 356.


All options are saved between uses of the Browser to the Browser profile (ibrs.ini).

---

### Manipulating Graphs

You can perform many actions on a graph. You can:

- Select Show Inheritance Graph from the Actions PullDown PullDown menu to get a large graph.
- Select Overview... from the View PullDown menu to get the overview window for navigating around large graphs.
- Drag the Zoom slider on the left side of the window up and down to zoom the graph in and out.
- Press Mouse Button 1 and drag it. This creates a dotted selection box. There is a PopUp for the selected area to Zoom, Print, etc. the selected area.
- Select Center from the **Graph** PopUp menu to center the currently selected node on the graph area display.

You can print the graph to a single page or across multiple pages. You can also print a selected zone in the graph.  For more information on printing graphs, see “Printing and Saving your Graphs” on page 361.

You can save graphs to a bitmap file, or copy them to the clipboard for including graphs in your own documentation.

Of course, there is Print, Save to a file, and Copy to the clipboard in the List Window too.



---

## **The Browser and WorkFrame**

The Browser is fully integrated with WorkFrame.

- You can browse a WorkFrame project.
- You can invoke Browser actions from the VisualAge Debugger, and perform browse actions on words highlighted in this tool.
- Other project actions appear in the **Project** PullDown.

## **Browser: A Tour**




## Chapter 27. Troubleshooting

This chapter helps you solve problems you may encounter while using the Browser.

---

### The Browser Won't Start

If you cannot start the Browser either by using the `ibrs` on the command line or by launching the **Browser** action from any VisualAge for C++ tools menubar, then delete the `ibrs.ini` file and start the Browser.

This file contains your user defined settings, such as color, size, placement, paths, etc. You can find this file in the directory where the operating system is installed, unless you specified a different location to store this file through the **Browser Settings** Paths Notebook page.  For more information on this dialog, see “Changing Paths Used by the Browser” on page 363.

---

### Error Loading a .EXE, .DLL, or .LIB file

If you are having problems loading an EXE, a DLL, or a LIB file, the file was either created using another vendor's compiler, or the **/Browse** linker option was not specified when it was created.


---

### Error Loading a .PDB File

Please contact your IBM service representative.

---

### Adding Files to the Load ► and Merge ► Menus Doesn't Work

You need to create an ASCII file called `brsmenutxt`.  See “Loading Files into the Browser” on page 368 or “Merging Files” on page 369 for instructions on how to create this file.

If you have already created this file, make sure that it is located in a directory in your `DPATH`. You may need to reboot if you just added the directory to the `DPATH`.

If you have the `brsmenutxt` file in a directory in the `DPATH`, but the files are still not displayed in the Load ► and Merge ► Cascade menus, make sure that there is a blank line at the end of the `brsmenutxt` file.

## Browser: Troubleshooting

---

### The Graph Zone Will Not Maximum Zoom

If you select a region of a graph and perform a Zoom in, but the select region does not fill the entire client area of the Graph window, this is not an error. The graph can only be zoomed to its maximum size allowed, it cannot zoom in beyond that. In other words, you have selected a region of the graph that is smaller than the maximum size of the Graph window client area.



## Chapter 28. Browser Fast-Path Keys and Menu Descriptions



This chapter lists the combinations of keys that you can use to perform Browser functions, the List and Graph window PullDown and PopUp menus, and the Object PopUp menus.

- “Fast-Path Keys”
- “PullDown Menus” on page 416
- “PopUp Menus” on page 424

---

### Fast-Path Keys

The following two lists outline the fast-path keys for using the Browser. The format is as follows: <key> or <key> - <key>.

**Note:** The letter keys are shown in uppercase for clarity. You do NOT have to press the **Shift** key unless this key is specifically mentioned.

#### Application-Provided Keys:

<b>F3</b>	Close the Browser session.
<b>F5</b>	Refresh the currently loaded Browser database file
<b>F6</b>	Perform the previous object-action pair.
<b>F7</b>	Expand all the items in the List window.
<b>F8</b>	Collapse all the items in the List window.
<b>Ctrl-C</b>	Centers the currently selected node on a graph
<b>Ctrl-E</b>	Edit the currently selected definition or file.
<b>Ctrl-F</b>	Initiate <b>Find</b> dialog to find text in the current window.
<b>Ctrl-G</b>	Graph all the base and derived classes for the currently selected class.
<b>Ctrl-H</b>	Show the documentation for the currently selected class or function.
<b>Ctrl-L</b>	List all members with inheritance for the currently selected class.
<b>Ctrl-N</b>	Find next instance of text in the current window.
<b>Ctrl-S</b>	Initiate <b>Search</b> dialog to search for text in the loaded database.
<b>Ctrl-Insert</b>	Copies the current window contents to the clipboard.
<b>Ctrl++</b>	Zoom in on a graph. (Use only the Numeric keypad)
<b>Ctrl--</b>	Zoom out on a graph. (Use only the Numeric keypad)
<b>Alt++</b>	Maximum zoom in on a graph
<b>Alt--</b>	Maximum zoom out on a graph

## Browser: PullDown Menus










### System-Provided Keys:

**Alt-F4**      Close window

---








## PullDown Menus

The PullDown menus can be found on both the List and Graph window, although some PullDowns are specific to each window.

-  “File PullDown Menu” on page 417.
-  “Edit PullDown Menu” on page 418.
-  “View PullDown Menu” on page 419.
-  “Actions PullDown Menu” on page 420.
-  “Options PullDown Menu” on page 421.
-  “Order PullDown Menu” on page 422.
-  “Windows PullDown Menu” on page 422.
-  “Project PullDown Menu” on page 423.
-  “Help PullDown Menu” on page 423.

## Browser: File Menu


### File PullDown Menu

Menu Item	Description	Window
<b>Load ►</b>	<p>Loads your program or Browser database files, or you can quickly load the classes that make up the VisualAge for C++ Open Class Library (User Interface Classes, Collection Classes, I/O Stream Classes, Complex Math Classes, Database Access Classes, Application Support Classes, and OLE Framework Classes).</p> <p> See “Loading Files into the Browser” on page 368 for more information on loading files into the Browser. See “Adding Menu Items to the Load ► and Merge ► Cascade menus” on page 393 for information on adding your own menu items.</p>	List, Graph
<b>Merge ►</b>	<p>Extends programs with additional controls or features from another program, or merge the classes that make up the VisualAge for C++ Open Class Library (User Interface Classes, Collection Classes, I/O Stream Classes, Complex Math Classes, Database Access Classes, Application Support Classes, and OLE Framework Classes).</p> <p> See “Merging Files” on page 369 for more information on merging files into the Browser. See “Adding Menu Items to the Load ► and Merge ► Cascade menus” on page 393 for information on adding your own menu items.</p>	List, Graph
<b>Refresh</b>	<p>Updates your current Browser database with the best source of data possible.  See “Updating the Browser Database” on page 392.</p>	List, Graph
<b>Save Graph As...</b>	<p>To save the Graph Area to a bitmap file.  See “Printing and Saving your Graphs” on page 361.</p>	Graph
<b>Save List As...</b>	<p>To save the contents of the List window or <b>List Area</b> of a Graph window to an ASCII file.  See “Printing and Saving your Lists” on page 350 or “Printing and Saving your Graphs” on page 361.</p>	List, Graph
<b>Print...</b>	<p>Prints the List window contents to the printer.  See “Printing and Saving your Lists” on page 350.</p>	List
<b>Print ►</b>	<p>Prints the Graph window contents to the printer (One Page..., Multiple Pages..., Client..., and Zone...).</p> <p> See “Printing and Saving your Graphs” on page 361.</p>	Graph

## Browser: Edit Menu

Menu Item	Description	Window
<b>New Window</b>	Creates another List window if launched from a List window, or another Graph window if launched from a Graph window. The new window will be blank.	List, Graph
<b>Copy Window</b>	Copies the current List or Graph window contents and puts them into a new List or Graph window.	List, Graph
<b>Exit Browser (F3)</b>	Ends your current Browser session. All List and Graph windows are closed.	List, Graph


## Edit PullDown Menu

Menu Item	Description	Window
<b>Find... (Ctrl-F)</b>	Launches the <b>Find</b> dialog. You can search the objects in the current window for the first instance matching the text string that you entered into the <b>Find</b> dialog.  See “Finding Objects in the Current Window” on page 371.	List, Graph
<b>Find Next (Ctrl-N)</b>	Searches the objects of the current window for the next instance of the text string that was last entered into the Find dialog. It does not search the entire Browser database. Note that the search begins from the current selection position in the window.	List, Graph
<b>Copy</b>	Copies the currently selected line to the clipboard.	List
<b>Copy All (Ctrl-Insert)</b>	Copies the entire contents of the List or Graph window to the clipboard.	List, Graph







## Browser: View Menu

### View PullDown Menu

Menu Item	Description	Window
<b>Overview...</b>	Launches a window which displays a miniature version of the current graph.  See “Getting a Graph Overview” on page 354.	Graph
<b>Zoom in (Ctrl-+)</b>	Increases the magnification of the current graph by approximately 10%.	Graph
<b>Zoom out (Ctrl--)</b>	Decreases the magnification of the current graph by approximately 10%.	Graph
<b>Max Zoom in (Alt-+)</b>	Increases the current graph to the maximum magnification.	Graph
<b>Max Zoom out (Alt--)</b>	Decreases the current graph to the minimum magnification.	Graph
<b>Center (Ctrl-C)</b>	Moves the currently selected node to the center of the graph area.	Graph
<b>Vertical</b>	Draws the graph with a vertical orientation. That is, the nodes are read from top to bottom.	Graph
<b>Horizontal</b>	Draws the graph with a horizontal orientation. That is, the nodes are read from left to right.	Graph
<b>Weighting ►</b>	<p>Adjusts where the nodes on the graph are displayed depending on the Vertical/Horizontal setting.</p> <p>Top - Aligns all the root nodes at the top/right of the graph.</p> <p>Center - Aligns the nodes around the center of the graph.</p> <p>Bottom - Aligns all the leaf nodes to the bottom/left of the graph.</p>	Graph




## Browser: Actions Menu

### Actions PullDown Menu

Menu Item	Description	Window
<b>Previous (F6)</b>	Returns to the last object-action pair performed in that window.  See “The History Window” on page 373.	List, Graph
<b>Search... (Ctrl-S)</b>	Launches the Search dialog in order to search the current Browser database for classes, functions, types, variables, and files.  See “Searching for Objects in the Entire Browser Database” on page 372.	List, Graph
<b>Show Inheritance Graph</b>	Displays the tree structure that represents how all the defined classes in the entire Browser database are related by inheritance.  See “Viewing Class Relationships” on page 386.	List, Graph
<b>Show Include File Graph</b>	Displays a tree structure that represents how all the source files in the current Browser database are related by the C and C++ include mechanism.  See “Viewing Call Chains” on page 387.	List, Graph
<b>List All Classes</b>	Creates an alphabetical list of all the classes defined or declared in the Browser database.	List, Graph
<b>List All Files</b>	Creates an alphabetical list of all the source files used to create the currently loaded Browser database.	List, Graph




## Browser: Options Menu

### Options PullDown Menu


Menu Item	Description	Window
<b>Fonts...</b>	Launches the <b>List Window Fonts</b> dialog to change the fonts used by the Browser.	List
<b>Node Fonts...</b>	Launches the <b>Graph Node Fonts</b> dialog in order to change the default text font used in the nodes.	Graph
<b>List Fonts...</b>	Launches the <b>Graph List Fonts</b> dialog in order to change the default text font used in the <b>Graph List Fonts</b> of the Graph window.	Graph
<b>List Window...</b>	Launches the <b>List window Settings</b> NoteBook. It has three pages: Settings, Colors, and Styles.  See “Changing the Default List Window Settings” on page 345.	List, Graph
<b>Graph Window...</b>	Launches the <b>Graph window Settings</b> NoteBook. It has four pages: Settings, Colors, Styles, and Bitmap.  See “Changing the Default Graph Window Settings” on page 356.	List, Graph
<b>Browser...</b>	Launches the <b>Browser Settings</b> NoteBook. It has two pages: Paths and Help Level.  See “Changing Browser Settings” on page 363.	List, Graph

## Browser: Order Menu • Browser: Windows Menu

### Order PullDown Menu

Menu Item	Description	Window
<b>Class</b>	Reorders the current objects in the list. The base classes are at the highest level, and class members are listed by type below.  See “Ordering the Contents of a Container View” on page 342.	List
<b>Access</b>	Reorders the current objects in the list with the access method at the highest level.  See “Ordering the Contents of a Container View” on page 342.	List
<b>Type</b>	Reorders the current objects in the list with the type of object at the highest level.  See “Ordering the Contents of a Container View” on page 342.	List

### Windows PullDown Menu

Menu Item	Description	Window
<b>History...</b>	Launches the History window. It lists the last 40 object-action pairs performed.  See “The History Window” on page 373.	List, Graph
<b>List 1</b> <b>List 2</b> <b>List 3</b> <b>List 4</b>	Gives you quick access to the List windows that you opened. If you do not have any List windows open, these menu items do not appear.	List, Graph
<b>Graph 1</b> <b>Graph 2</b> <b>Graph 3</b> <b>Graph 4</b>	Gives you quick access to the Graph windows that you opened. If you do not have any Graph windows open, these menu items do not appear.	List, Graph

## Browser: Project Menu • Browser: Help Menu

### Project PullDown Menu

Menu Item	Description	Window
<b>IBM WorkFrame actions</b>	Items in this menu are dependent on what you have installed and how you launched the Browser.	List, Graph

### Help PullDown Menu



Menu Item	Description	Window
<b>Help Index</b>	Launches the index for the Browser online help.	List, Graph
<b>General Help</b>	Launches the online help panel for either the List window or Graph window depending on which window you chose this item from.	List, Graph
<b>Using Help</b>	Launches help information for using the Information Presentation Facility (IPF).	List, Graph
<b>How Do I...</b>	Launches the <b>How Do I...</b> information which provides step-by-step instructions on how to perform tasks using the Browser.	List, Graph
<b>VisualAge for C++ Documentation Cascades</b>	Launches the various online documentation for the VisualAge for C++. These cascades are populated with the documents depending on what you have installed.	List, Graph
<b>Product Information</b>	Provides information about this release of the Browser.	List, Graph

## Browser: PopUp Menus •Browser: Window PopUp Menus

---



### PopUp Menus

The Browser has two types of PopUp menus:

-  “PopUp Menu Items for List and Graph Windows.”
-  “Object PopUp Menu Items” on page 425.

### PopUp Menu Items for List and Graph Windows

You can access the List and Graph window PopUps by clicking Mouse Button 2 on the background area of the window. The following items will appear, depending on the window you accessed the PopUp from:

Menu Item	Description	Window
<b>Show Inheritance Graph</b> <b>Show Include File Graph</b> <b>List All Classes</b> <b>List All Files</b>	 See “Actions PullDown Menu” on page 420.	List, Graph
<b>Expand All</b>	Expand all items in the List window container view.	List (container view)
<b>Collapse All</b>	Collapse all items in the List window container view.	List (container view)
<b>Overview...</b> <b>Zoom in</b> <b>Zoom out</b> <b>Max Zoom in</b> <b>Max Zoom out</b> <b>Center</b> <b>Vertical</b> <b>Horizontal</b> <b>Weighting ►</b>	 See “View PullDown Menu” on page 419.	Graph

## Browser: Object PopUp Menus

### Object PopUp Menu Items

You can access the Object PopUps by clicking Mouse Button 2 on any program element displayed in the List or Graph windows. The following items will appear, depending on which window you accessed the PopUp from:

Menu Item	Description	Object
<b>Collapse</b>	Collapse the currently selected object entirely if it has a minus icon (-) next to it.	class, label
<b>Edit Definition (Ctrl-E)</b>	Launches the VisualAge for C++ Editor and loads the file containing the definition of the currently selected object. The editor is positioned to the first line of the definition of this object.	class, function, type, variable
<b>Edit File (Ctrl-E)</b>	Launches the VisualAge for C++ Editor and loads the file.	file
<b>Expand</b>	Expands the currently selected object entirely if it has a plus icon (+) next to it.	class, label
<b>Expand Typedef</b>	Creates a list which contains successive expansions of a typedef, until it contains only fundamental types.	type
<b>Graph All Base &amp; Derived Classes (Ctrl-G)</b>	Creates an inheritance graph of all the base classes of the selected class, and all the classes that derive from the selected class, in one relationship graph. These include both direct and indirect base and derived classes.	class
<b>Graph All Base Classes</b>	Creates an inheritance graph of all the base classes of the selected class. These include both direct and indirect base classes.	class
<b>Graph All Derived Classes</b>	Creates an inheritance graph of all the classes that are derived from the selected class. These include both direct and indirect derived classes.	class
<b>Graph Immediate Derived Classes</b>	Create a Graph window which displays one level of derived classes for this class.	class
<b>Graph All Callers &amp; Calleees</b>	Creates a graph of all the functions that call the currently selected function and the functions that are called by the currently selected function. These include the functions that the selected function calls and is called from, both directly or indirectly.	function

## Browser: Object PopUp Menus

Menu Item	Description	Object
<b>Graph All Callers</b>	Creates a graph of all the functions that call the currently selected function. These include the functions which call the selected function directly, and those functions that call it indirectly through other functions.	function
<b>Graph All Callees</b>	Creates a graph of all the functions that the currently selected function calls. These include the functions that the selected function calls directly or indirectly.	function
<b>Graph Immediate Callers &amp; Callees</b>	Creates a graph of all the functions that either call the selected function directly or is called from directly.	function
<b>Graph All Includers &amp; Includees</b>	Creates a graph of all the files included by the currently selected file and all the files that include the currently selected file. These include both direct and indirect inclusion.	file
<b>Graph All Includers</b>	Creates a graph of all the files that include the currently selected file. These include both direct and indirect inclusion.	file
<b>Graph All Includees</b>	Creates a graph of all the files that are included by the currently selected file. These include both direct and indirect inclusion.	file
<b>List Class Members with Inheritance</b>	Creates a List window container view of all the members of the classes and base classes for the class that the selected function is a member of.	function
<b>List Defined Objects</b>	Creates a list of all the Browser objects which are defined in the selected file.	file
<b>List Friends</b>	Lists all the friends of the currently selected class. A friend of a class is a function that has been granted access to the private members of the class. A friend class obtains access to private members for all its member functions. The result of this action is a List window container view with functions and classes as the two labels in the container.	class
<b>List Friendships</b>	Lists all the friendships that are defined for the currently selected class or function. You can grant friendships to either a single function or member function at a time, or to all the member functions of a class at once. This action results in a list of all those classes that have granted the selected class or function friendship.	class, function



## Browser: Object PopUp Menus

Menu Item	Description	Object
<b>List Immediate Callers &amp; Callees</b>	Creates a list of all the functions that directly call or are directly called by the selected function.	function
<b>List Instantiations</b>	Creates a list of the instantiations of the selected class or function template.	class, function
<b>List Implementing Files</b>	Create a list of files that contain definitions for the class, and for any of the members of the class.	class
<b>List Members with Inheritance (Ctrl-L)</b>	Creates a List window container view of all the members for the selected class and its base classes.	class
<b>List Overriding Derived Classes</b>	Creates a list of all the derived classes of the class of which this function is a member which have a member function that overrides this function.	function
<b>List Possible Exceptions Thrown</b>	Creates a list of all the possible types of exceptions that could be thrown by the selected function, or by any function that this function calls.	function
<b>Show Documentation (Ctrl-H)</b>	Launches the help panel in the VisualAge for C++ class library documentation for the currently selected object.	class, function

## **Browser: Object PopUp Menus**

---

## Part 7. Performance Execution Trace Analyzer

**Note:** Additional information is available in the Performance Analyzer online Help facility. To access this information, select **General help** from the **Help** menu on any Performance Analyzer window. After the Help facility starts, you can view the table of contents by selecting **Contents** from the **Options** menu.

For an index of Performance Analyzer topics, select **Help index** from the **Help** menu on any Performance Analyzer window.

You may find that the hypertext links in the online Help are easier to use.

## **Performance Execution Trace Analyzer**



## Chapter 29. Introducing the Performance Execution Trace Analyzer

The IBM VisualAge for C++ for Windows, Version 3.5 Performance Execution Trace Analyzer is an application that helps you understand and improve the behavior of C and C++ programs.

The Performance Analyzer traces the execution of a program and creates a trace file. The trace file contains trace analysis data that can be displayed in diagrams. Using these diagrams, you can improve program performance, examine occurrences that produce faults, and in general, understand what happens when a program runs.

The Performance Analyzer does not replace static analyzers or debuggers, but it can complement them by helping you understand aspects of the program that would otherwise be difficult or impossible to see.

For instance, with the Performance Analyzer you can:

### **Time and tune programs**

The Performance Analyzer time stamps each trace event (a function call or return) using a high resolution clock (about 838 nanoseconds per clock tick). As a result, the trace file contains a detailed record of when each traced function was called and when it returned.

The trace data also shows how long each function executed, which helps you find hot spots.

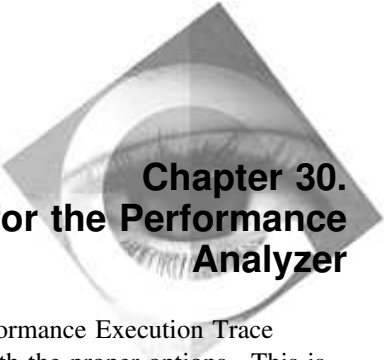
### **Locate program hangs and deadlocks**

The Performance Analyzer provides a complete history of events leading up to the point where a program stops. You can view the function call stack from anywhere in the program.

### **Trace multithreaded interactions**

When multithreaded programs are traced, you can look at the sequencing of functions across threads in some of the diagrams. This highlights problems within critical areas of the program.

## **Performance Execution Trace Analyzer**



## Chapter 30. Preparing Your Program for the Performance Analyzer

Before you create a trace file and begin using the Performance Execution Trace Analyzer, you *must* compile and link your program with the proper options. This is described in “Compiling and Linking Your Program.”

The Performance Analyzer provides several ways to customize trace files. If you want to customize a trace file, you may have to complete some steps *before you compile and link your program*. For more information and instructions, see the following topics:

- “Tracing Dynamic Link Libraries (DLLs)” on page 434
- “Tracing System Calls” on page 434
- “Creating User Events in Your Program” on page 435
- “Starting and Stopping the Performance Analyzer from Your Program” on page 436.

---

### Compiling and Linking Your Program

You *must* compile and link your program with the proper options before you create a trace file and analyze it with the Performance Analyzer.

#### Compiling

When compiling your program, use the following options:

- /Gh** Includes the profile hooks (code added at the start and end of each function) that allow the Performance Analyzer to monitor your executable.
- /Ti** Includes debugging information in the compiled object file.

#### Linking

You must link the CPPWPA3.OBJ object file into your program.

When linking your program, use the following options:

- /DE** Instructs the linker to include debug information in the executable (EXE) or dynamic link library (DLL) file.
- /NOE** Instructs the linker not to search for symbols in the extended dictionaries of the libraries being linked.

## Preparing Your Program for the Performance Analyzer

### Example

The following example shows how to compile and link a program called *SAMPLE.EXE* for use with the Performance Analyzer. The required object file is highlighted.

Compile:

```
icc /c /Ti /Gh sample.cpp
```

Link:

```
ilink /DE /NOE sample.obj cppwpa3.obj
```

---

## Tracing Dynamic Link Libraries (DLLs)

You can trace statically or dynamically linked DLLs using the Performance Analyzer. Compile and link any DLL to be traced. For instructions, see “Compiling and Linking Your Program” on page 433.

You can trace DLLs without tracing the main program that calls the DLLs.

The tracing of load-on-call DLLs occurs automatically; it cannot be suppressed. Also, you cannot set triggers, or enable or disable functions in dynamically loaded DLLs.

---

## Tracing System Calls

If you want to trace system API calls, specify the following Performance Analyzer libraries *before* the system libraries in your link statement. You can link one or more libraries.

### Notes:

1. Output produced by *cout* stream objects is stored in the operating system’s buffer and is not shown until tracing has stopped. System API calls for the *cout* stream are not displayed in the trace file.
2. It is not possible to trace events in the KERNEL32 intercept library *only*. The Performance Analyzer looks for at least one event from your program before logging these call events. If you link the *\_KERNEL.LIB* library, compile your program with the */Gh* and */Ti* options to include events from your program in the trace file.
3. If you compile your source with the */Gd-* switch, the trace file will contain all of the system calls called by your code *and* the system calls called by VisualAge C++ runtime calls. To avoid tracing calls made by runtime calls, use the */Gd+* switch.



## Preparing Your Program for the Performance Analyzer

The Performance Analyzer libraries are as follows. Each library listed also has an associated DLL.

- `_KERNEL.LIB`
- `_USER32.LIB`
- `_GDI32.LIB`

**Important:** The order in which these libraries are specified in the link statement is critical. If the replacement libraries do not precede the system libraries in the link statement, the Performance Analyzer will not interpret and trace the API calls.

For a list of traced functions in these DLLs, see the online Help topic “Traced Functions in Intercept Libraries.”

---

### Creating User Events in Your Program

The `CPPWPA3.OBJ` file contains an entry point called *PERF* that accepts calls from the program you are tracing. Calls to the *PERF* entry point at execution time are referred to as user events. User events cause text strings to be inserted into the trace file.

To add a call to the *PERF* entry point:

1. Declare a prototype for the *PERF* entry point.

If you add a user event to your program source file, you must also include a prototype for the *PERF* entry point.

For C and C++ programs, the prototype will be inserted for you when you compile your program if you place the following statement at the beginning of your source file:

```
#include <iperf.h>
```

**Note:** If you want to insert the prototype yourself, the prototypes for C and C++ programs are as follows:

#### *C Prototype*



```
void PERF (const char* string);
```

#### *C++ Prototype*



```
extern "C" {void PERF(const char* string);}
```

## Preparing Your Program for the Performance Analyzer

2. Add a call to the entry point everywhere you want a user event generated.

The following is an example of a call to the PERF entry point:



```
PERF (string);
```

where:

*string* is an ASCIIZ string.

When such a call is made, the string is placed in the trace file. You can see the string in the **Call Nesting**, **Statistics**, and **Time Line** diagrams.

**Warning:** The *string* must exist in storage when your program terminates. If the *string* exists in automatic storage on the stack or storage defined in a dynamically loaded DLL, the *string* will appear in the trace file, but the text may not appear as it was defined. (The *string* can be allocated on the heap if its contents are not deleted when your program terminates.)

---

## Starting and Stopping the Performance Analyzer from Your Program

The CPPWPA3.OBJ file contains entry points called *PerfStart* and *PerfStop* that accept calls from the program you are tracing. Calls to the *PerfStart* and *PerfStop* entry points cause the Performance Analyzer to start and stop tracing, respectively. By putting these calls into your source code, you can control precisely when the Performance Analyzer starts and stops recording events during program execution. To turn tracing:

**On** Call the *PerfStart* entry point.

**Off** Call the *PerfStop* entry point.

### Notes:

1. If tracing is already on, calling *PerfStart* has no effect. If tracing is already off, calling *PerfStop* has no effect.
2. You can also start and stop tracing with the **Trace on** and **Trace off** push buttons on the **Application Monitor** window.

## Preparing Your Program for the Performance Analyzer

You can place calls to the PerfStart and PerfStop entry points anywhere in your program, even in different functions, object modules, or DLLs.

To add calls to the PerfStart and PerfStop entry points:

1. Declare a prototype for the PerfStart and PerfStop entry points.

For C and C++ programs, the prototypes will be inserted for you when you compile your program if you place the following statement at the beginning of your source file:

```
#include <iperf.h>
```

**Note:** If you want to insert the prototypes yourself, the prototypes for C and C++ programs are as follows:

### *C Prototype*



```
void PerfStart (void);  
void PerfStop (void);
```

### *C++ Prototype*



```
extern "C" {void PerfStart (void);}  
extern "C" {void PerfStop (void);}
```

2. Add a call to the appropriate entry point everywhere you want to start or stop tracing.

The following example shows how calls to the PerfStart and PerfStop entry points could be placed in your program:



```
.  
. .  
PerfStop(); // turn off tracing here  
. .  
PerfStart(); // resume tracing here
```

## Preparing Your Program for the Performance Analyzer

---

### Understanding Overhead Time

When you compile and link your program, the compiler generates hooks (code added at the start and end of each function) that enable the Performance Analyzer to intercept trace events. The starting and ending times for each trace event are recorded in the trace file.

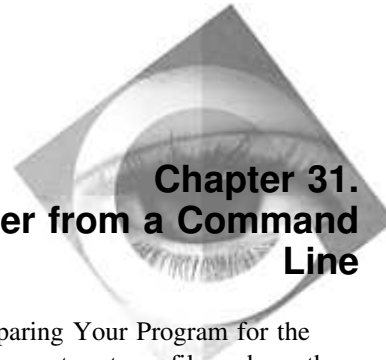
These hooks cause a small monitoring function to be called instead of the program's callee function. The monitoring function time stamps the event and then calls the program's callee function.

The monitoring function is run in the program's address space, thereby avoiding the high overhead of an operating system context switch when events are recorded. As a result, it does not significantly affect the program's execution performance. However, the monitoring function does take a small amount of time to execute. In order to compensate for this additional time introduced by the monitoring function, the diagrams adjust the timings appropriately.

The Performance Analyzer dynamically determines how much time it takes to execute the monitoring function by internally calling it several times and computing an average prior to executing the program.

**Note:** It is recommended that you shut down other programs on your desktop so they will not interfere with the Performance Analyzer's timings.

If the program is run stand-alone, it should run at or near the same speed as the same program compiled without the profile hooks because trace events are not recorded.



## Chapter 31. Starting the Performance Analyzer from a Command Line

After completing the instructions in Chapter 30, “Preparing Your Program for the Performance Analyzer” on page 433, you are ready to create a trace file and use the Performance Analyzer to analyze your program.

The command you enter to start the Performance Analyzer depends on which of the following you want to do first:

- Trace an executable
- Analyze an *existing* trace file
- Display the Performance Analyzer’s main control window.

---

### Tracing an Executable


- If you have an executable you want to trace, you can start the analyzer from a command line, a command file (.CMD), or a batch file (.BAT) by entering:

```
iperf myprog parms
```

where:

**myprog** Represents an executable file name. This is optional.  
**parms** Represents executable parameters. These are optional.

**Note:** The first time you start the Performance Analyzer, a profile window appears and prompts you to specify where the Performance Analyzer’s profile file should be located. If you want the default, press the **OK** push button.

 See “Performance Analyzer - Specify Profile Location Window” on page 457 for more information.

- You can also include the `/go` option in the command:

```
iperf /go myprog parms
```

where:

**/go** Executes your program, creates a trace file, and then exits the Performance Analyzer. This option is useful if you have several programs (requiring no manual intervention) that you want to run in succession from a command file. This is optional.

<b>myprog</b>	Represents an executable file name. This is required when the <i>go/</i> option is specified.
<b>parms</b>	Represents parameters associated with the <i>myprog</i> executable. These are optional.

---

## Analyzing an Existing Trace File

If you want to start analyzing a trace file that you have already created, you can start the Performance Analyzer from a command line, a command file (.CMD), or a batch file (.BAT) by entering:

```
iperf /x myprog.trc
```

where:

**/x** Represents one or more of the following analyzer options. These options cause the trace file to be displayed in their respective diagrams. Once you are familiar with the Performance Analyzer application, you can quickly open the diagrams by entering as many of these options as you want in your startup command. This is optional.

**/cn** Displays the trace file in the **Call Nesting** diagram.

**/ed** Displays the trace file in the **Execution Density** diagram.

**/cg** Displays the trace file in the **Dynamic Call Graph**.

**/ss** Displays the trace file in the **Statistics** diagram.

**/tl** Displays the trace file in the **Time Line** diagram.

**myprog.trc** Represents a trace file name.

---

## Displaying the Performance Analyzer's Main Control Window

If you enter the following command, the analyzer's main control window, the **Performance Analyzer - Window Manager** window appears:

```
iperf
```

From this window, you can start either tracing an executable or analyzing an existing trace file.

 See Chapter 34, "Creating a Trace File" on page 445 to continue.

## Starting the Performance Analyzer from WorkFrame

Before you start the Performance Analyzer from the WorkFrame environment, you must:

1. Create a project for the program you want to analyze.  
**Note:** For information on creating a project, refer to the WorkFrame documentation.
2. Open a project folder in the WorkFrame window.
3. Compile and link your program with Performance Analyzer options.  
**Note:** This is described in “Compiling and Linking Your Program” on page 433.
4. Highlight an object that represents an executable file or a trace file.
5. Click mouse button 2 on the highlighted object to display a pop-up menu.
6. Select **Analyze**.







## Chapter 33. Exiting the Performance Analyzer

If you want to exit the Performance Analyzer, and are *not* in the process of creating a trace file, do the following:

1. Select the **Exit the Performance Analyzer** choice from one of the following menus:
  - **File** menu on the **Performance Analyzer - Window Manager** window
  - **Application** menu on the **Trace Generation** window
  - **Trace file** menu on any of the diagrams.
2. Select **Yes** when prompted.

If you want to exit the Performance Analyzer while a trace file is being created, do the following:

1. Click on the **Stop** push button on the **Application Monitor** window.
2. Click on the **Cancel** push button on the **Analyze Trace** window.
3. Select the **Exit the Performance Analyzer** choice from the **File** menu on the **Performance Analyzer - Window Manager** window.
4. Select **Yes** when prompted.





## Chapter 34. Creating a Trace File

After compiling and linking your program, you can start the Performance Analyzer and create a *trace file*. A *trace file* contains a chronological sequence of events (function calls and returns) that occur during the execution of your program.

By analyzing the trace file, you can learn about your program's structure, locate and diagnose problems, and pinpoint ways to improve performance. The Performance Analyzer provides five diagrams in which you can analyze the trace file. Each diagram presents a different view of the trace file to give you an overall idea of how your program performs. The diagrams are as follows:

- Call Nesting
- Dynamic Call Graph
- Execution Density
- Statistics
- Time Line

**Note:** Before creating a trace file, you must prepare your program for use by the Performance Analyzer. For more information, see Chapter 30, “Preparing Your Program for the Performance Analyzer” on page 433.

To create a trace file:

1. Click on the **Create Trace** push button in the **Performance Analyzer - Window Manager** window.
2. Type the full path name and the file name of the program you want to trace in the **Program Name** entry field. If the program is in your current directory, you do not have to type the path name.

**Note:** If you are not sure where the file is located, select the **Find** push button.

3. Type any parameters that you want to pass to your program in the **Program Parameters** entry field.

**Note:** This entry field is optional.

4. If you want the trace file to have a different path and file name than the defaults, type a path and file name in the **Trace File Name** entry field.

The default path name is the directory where your program resides. The default trace file name is *myprog.trc*, where *myprog* is the name of the program you are tracing.

**Note:** This entry field is optional.

5. Type any comments that you want to make about your trace in the **Trace File Description** entry field.

**Note:** This entry field is optional.

6. Select the **OK** push button. The **Trace Generation** window appears.
7. Select the **Trace** push button in the **Trace Generation** window.

Your program begins executing. When your program ends, the **Analyze Trace** window is displayed.

8. Click on the check box next to each diagram in which you want to view the trace file.

**Note:** If you open your trace file in a diagram and find that the file does not contain any trace events, your application may have been unable to locate CPPWPA3.DLL when you created the trace file. To correct this, change your path to point to the location of CPPWPA3.DLL (probably in the DLL subdirectory where you installed VisualAge for C++), and then recreate your trace file.



## Chapter 35. Creating a Customized Trace File

By default, the Performance Analyzer generates event information for every function executed that has been compiled and linked with the proper options. However, this sometimes causes the trace file to become large and difficult to manage. Furthermore, you may not want or need all of this information.

To allow you to control the size of your trace file, the Performance Analyzer provides parameters that you can set prior to running your program.

The following parameters affect the size of the trace file:

- Enabled or disabled state of components (explained on page 447)
- Call depth setting for each thread (explained on page 449)
- Time stamp setting (explained on page 449)
- Trigger settings (explained on page 449)
- Buffer wrap setting (explained on page 450).

Although they do not affect the size of your trace file, the following items can also be customized:

- Trace file name. The default file name is *myprog.trc*, where *myprog* is the name of the program you are tracing.
- Trace file description. A description can make a trace file easier to identify, especially when you create more than one trace file from the same program and use different options for each trace. The description is displayed in the **Status Area** of any open diagram.

For more information on specifying a different trace file name or attaching a description to a trace file, see the online Help topics “Name Trace File Choice” and “Unique Trace File Name Choice”.

---

### Enabling and Disabling Components

Your program’s components are listed on the **Trace Generation** window.

**Note:** A component can be an executable (EXE), a dynamic load library file (DLL), an object file (OBJ), or a function. EXEs and DLLs contain object files, and object files contain functions. To view or hide components in the window, click on the plus or minus icon to expand and contract EXEs, DLLs, and OBJ files.

The Performance Analyzer’s default is to enable all components that have been compiled and linked with the proper options. When a component is enabled, data for

that component will be included in the trace file when the component is executed. When the component is disabled, no data is recorded in the trace file when the component is executed.

Trace files containing data for all enabled components can sometimes become large and difficult to manage. You can limit the amount of data collected in your trace file by selecting specific components to enable and disable using the **Edit** menu on the **Trace Generation** window.

When enabling and disabling components, remember the following.


- When you disable:
  - An EXE, a DLL, or an OBJ file, the Performance Analyzer disables all functions within the selected file and removes any triggers set on functions within the file.
  - A function, the Performance Analyzer removes any trigger set on it.
- When you enable:
  - An EXE, a DLL, or an OBJ file, the Performance Analyzer enables all functions within the selected file.
- When you set a trigger on a disabled function, the Performance Analyzer enables the function.

You can enable or disable a component in any of the following ways:

- Click on the file name or icon of the component you want to enable or disable. Then select the appropriate enable or disable choice from the **Edit** menu on the **Trace Generation** window.
- Double-click on the file name or icon of the component you want to enable or disable.
- Click mouse button 2 on the file name or icon of the component you want to enable or disable. Then select the appropriate enable or disable choice from the pop-up menu.

**Note:** If the icon next to a component is:

- Green (with no slash mark), the component is *enabled*.
- Red (with a slash mark), the component is *disabled*.
- White, the component cannot be traced.

 For a description of the menu choices you can use to enable and disable components, see “Menu Bar Summary” on page 464 or the Performance Analyzer online Help facility.

---

## Selecting the Call Depth for Each Thread

You can specify the nesting depth that you want to trace in order to isolate an area of interest and reduce the amount of trace data.

Select **Call depth** from the **Options** menu on the **Trace Generation** window to select the nesting depth of function calls or to specify threads that you want to include in or exclude from the trace file. When the **Call Depth** window is displayed, you can select as many as 64 threads with a maximum nesting depth of 128 for each thread. The default is to have all threads selected with the maximum depth of 128.

---

## Using Time Stamps

Select **Time stamp events** from the **Options** menu on the **Trace Generation** window to choose whether to time stamp events during the trace analysis.

While logging events, the Performance Analyzer adds a small amount of overhead time to the normal execution time of the program through buffer flushing and writing to disk, and through the monitoring of events. Although the time added is negligible, the Performance Analyzer attempts to remove it from the timings reported in the diagrams. Therefore, the timings you see in the diagrams represent your program's actual execution time as nearly as possible.

If you choose to create a trace file without time stamps:

- You can only view the trace file in the **Dynamic Call Graph**, **Call Nesting**, and **Statistics** diagrams.
- All nodes in the **Dynamic Call Graph** will be the same size and color.
- Time data (such as execution time and time on stack) will not be present in either the diagrams or their related dialogs.

---

## Setting and Removing Triggers

A trigger turns tracing on when the function is called and then turns tracing off when it returns. By setting triggers to start or stop tracing at selected points in your program, you can control the size of your trace file. You can set and remove triggers on functions. The Performance Analyzer allows you to set multiple triggers.

You can set and remove triggers from the **Edit** menu and the **Function** pop-up menu on the **Trace Generation** window. Remember the following when setting triggers:

- If triggers are set, the Performance Analyzer starts tracing when it reaches the triggered function and continues tracing until it receives a return from that function. If no triggers are set, the Performance Analyzer traces all enabled components.

- If a trigger function is nested within another trigger function, tracing is turned off only after the outer function returns.
- A function that has a trigger set on it has the letter *T* in the icon next to its name in the **Trace Generation** window.
- If you set a trigger on a disabled function, the Performance Analyzer enables the function.
- If you disable a function, the Performance Analyzer removes any trigger set on it.
- If you disable an EXE, a DLL, or an object file, the Performance Analyzer disables all functions within the selected file and removes any triggers set on functions within the file.

---

## Enabling and Disabling Buffer Wrap

During a trace analysis, the Performance Analyzer and your program share memory with the trace buffer. The trace buffer allows the Performance Analyzer to log events that are running in the address space of the program.

When **Buffer wrap** is enabled, the Performance Analyzer overwrites older events in the buffer with newer ones. Since the buffer is flushed only when the program ends, the trace file is smaller, but some trace data is lost. The default setting for **Buffer wrap** is *disabled*.

If **Buffer wrap** is disabled and the trace buffer becomes full, the Performance Analyzer:

1. Pauses the program
2. Time stamps the start of the buffer flush
3. Writes the events in the buffer to the trace file
4. Time stamps the end of the buffer flush
5. Resumes the program.

**Note:** Time stamping the buffer flush allows the diagrams to remove the buffer-flushing overhead time.

You can select **Buffer wrap** from the **Options** menu on the **Trace Generation** window to enable or disable buffer wrapping. A check mark appears next to the choice when it is enabled. The check mark *does not* appear when the choice is disabled.

---

## Naming the Trace File

The default file name for a trace file is *myprog.trc*, where *myprog* is the file name of the program you are tracing. When you run several traces of the same program, the **Name trace file** choice lets you name each trace file and describe what you did differently for each trace.



For example, if you disable an object file for the first trace and disable time stamps for the second trace, you could name the first trace file *TRACE1* and enter *Disabled object file* for its description. Likewise, you could name the second trace file *TRACE2* and enter *Disabled time stamps* for its description.

To name a trace file, select **Name trace file** from the **Options** menu on the **Trace Generation** window. In the **Name Trace File** window, you can enter your own trace file name and a short description.

A trace file's description is displayed in the **Status Area** of any open diagram.



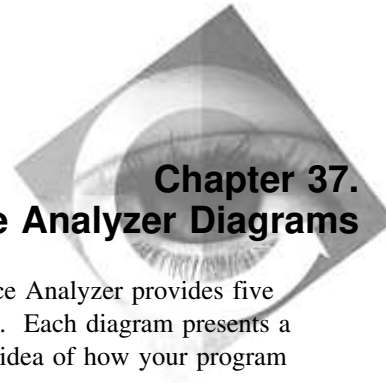


## Chapter 36. Saving Trace File Settings

Trace settings (settings that determine how a program is traced) that you enter are saved for the current session. You can save some trace settings for subsequent sessions by selecting **Options > Settings > Save** from most Performance Analyzer windows and any Performance Analyzer diagram.

For information about specific settings saved, see “Save Choice” in the Performance Analyzer Help facility.





## Chapter 37. Using the Performance Analyzer Diagrams

After you have created your trace file, the Performance Analyzer provides five diagrams in which you can view and analyze the data. Each diagram presents a different view of the trace file to give you an overall idea of how your program performs.

The following list contains a description of each diagram and the icon the Performance Analyzer uses to represent each diagram throughout the application:

### Icon/Diagram

### Description



#### Call Nesting

Useful for diagnosing specific performance problems. For example, you might use this diagram to see what led up to a thread switch.

The **Time Line** diagram is useful in correlation with this diagram. For example, you can mark a function call in this diagram and then correlate it to the **Time Line** diagram. The correlated **Time Line** diagram shows you when the function call occurred and how long it lasted.



#### Dynamic Call Graph

Useful for diagnosing overall performance problems. This diagram shows performance problem areas by color and size. Components (functions, classes [if you are analyzing a C++ program], or executables) that consume the most execution time will appear as large red rectangles. Components that are called excessively will have a red arc attached to them.



#### Execution Density

Useful for showing trends of program execution. This diagram displays trace data chronologically from top to bottom as thin horizontal lines of various colors in different columns.



#### Statistics

Useful for diagnosing overall performance problems. This diagram provides a textual report of execution time by component type: function,

class (only if you are analyzing a C++ program), or executable. Components are sorted by execution time, with the component consuming the most time appearing at the top of the diagram by default.

This diagram is also useful for determining which user functions to inline.



### Time Line

Useful for diagnosing specific performance problems. For example, you might use this diagram to determine why a window takes a long time to paint.

The **Call Nesting** diagram is very useful in correlation with this diagram. For example, you can mark a point in time in this diagram, and then correlate it to the **Call Nesting** diagram. The correlated **Call Nesting** diagram shows you the function name associated with the time you marked in the **Time Line** diagram.

---

## Opening a Trace File in a Diagram

To open a trace file in any diagram, use any of the following methods:

- Click on the **Analyze Trace...** push button in the **Performance Analyzer - Window Manager** window, and then, in the **Analyze Trace** window, enter a trace file name and select one of the diagram check boxes.
- Double-click on the file name or icon of a trace file in the **Performance Analyzer - Window Manager** window, and then select one of the diagram check boxes in the **Analyze Trace** window.
- Click mouse button 2 on the file name or icon of a trace file in the **Performance Analyzer - Window Manager** window, then select a diagram from the **Trace File** pop-up menu.
- From the **Trace File** menu of an open diagram, select **Open as** and then select a diagram from the cascaded menu.
- From any open diagram, click the appropriate button in the tool bar.

**Note:** If you open your trace file in a diagram and find that the file does not contain any trace events, your application may have been unable to locate CPPWPA3.DLL when you created the trace file. To correct this, change your path to point to the location of CPPWPA3.DLL (probably in the DLL subdirectory where you installed VisualAge for C++), and then recreate your trace file.

## Chapter 38. Introducing the Performance Analyzer Windows

The following pages briefly describe the Performance Analyzer windows. For more complete information, see the Performance Analyzer online Help facility. The primary Performance Analyzer windows are:

- Performance Analyzer - Specify Profile Location Window
- Performance Analyzer - Window Manager Window
- Create Trace Window
- Trace Generation Window
- Application Monitor Window
- Analyze Trace Window.

---

### Performance Analyzer - Specify Profile Location Window

When you start the Performance Analyzer for the first time, the **Performance Analyzer - Specify Profile Location** window appears.

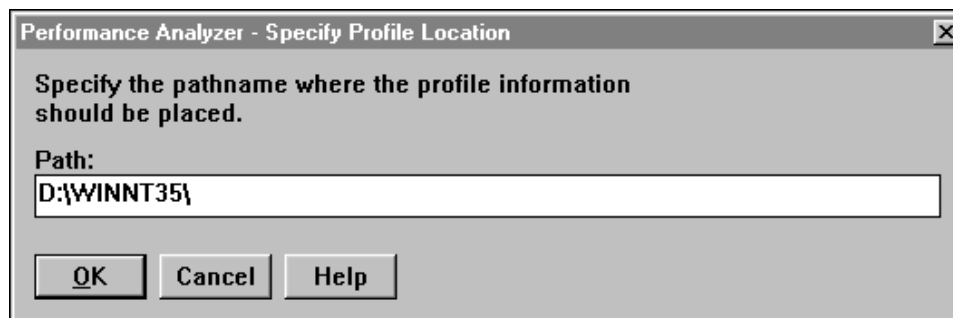


Figure 139. Performance Analyzer - Specify Profile Location Window

This window prompts you to type the path name where you want to store the *IPERF.INI* file. The *IPERF.INI* file stores your session settings. The default path name is the root operating system directory. If you want to store the *IPERF.INI* file in a drive and directory other than the default, type the full path name in the **Path** entry field, and then select **OK**. The *IPERF.INI* file is created in the directory you specified.

---

## Performance Analyzer - Window Manager Window

The **Performance Analyzer - Window Manager** window is the Performance Analyzer's main control window and is always displayed while the Performance Analyzer is running. Once you have properly compiled and linked your program and started the Performance Analyzer, you can start most functions from this window, including creating and analyzing trace files.

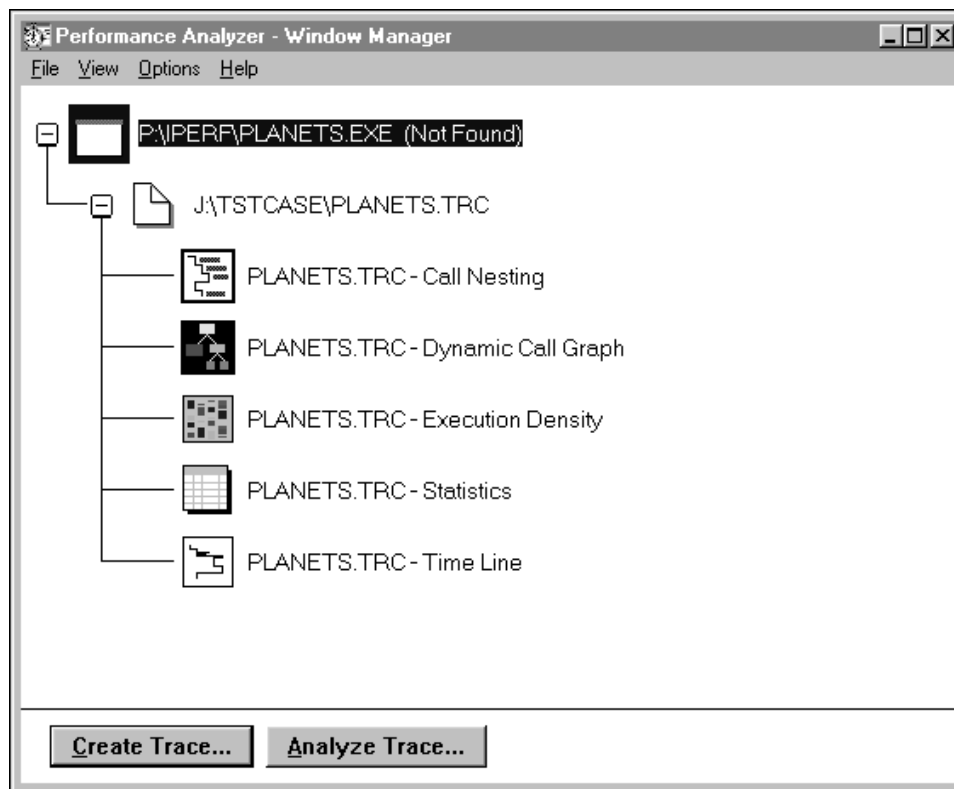


Figure 140. Performance Analyzer - Window Manager Window

When you view a trace file, this window lists the file names of your executable, your trace file, and each diagram in which the trace file is displayed.



## Areas of the Performance Analyzer - Window Manager Window

The following topics describe the areas of the **Window Manager** window.

### Menu Bar Summary

The menu choices in the **Window Manager** window are as follows:

#### File

From this menu, you can select:

##### Create Trace

Displays the **Create Trace** window, which lets you start creating a trace file for your program.

##### Analyze Trace

Displays the **Analyze Trace** window, which lets you open a diagram and start analyzing your program.

##### Exit the Performance Analyzer

Lets you end the Performance Analyzer application.

#### View

From this menu, you can select:

##### Tree lines

Displays tree lines on the **Performance Analyzer - Window Manager** window.

##### Show icons

Displays icons that identify file types on the **Performance Analyzer - Window Manager** window.

##### Remove all windows

Removes and closes all open diagrams from your screen.

#### Options

From this menu, you can select:

##### Font...

Displays the **Font** window, which lets you change the font, font style, and font size for the windows.

##### Quick exit

Controls whether the Performance Analyzer provides a confirmation prompt upon exit.

##### Search paths

When you are working in the WorkFrame environment, this choice displays a window in which you can specify where the Performance Analyzer can locate source files for editing.

##### Unique trace file name

Gives each trace file a different name, which allows you to save several trace files created from the same program.

##### Settings >

Displays a cascaded menu that lets you select either of the following:

**Project (only available when working in the WorkFrame environment)**

This menu appears on the **Performance Analyzer - Window Manager** window when you start the Performance Analyzer from within the WorkFrame environment.

WorkFrame actions that can be launched from the **Performance Analyzer - Window Manager** window will also appear in this menu.

**Help** This menu provides choices that display various types of Help information. From this menu, you can select:

**Help index**

Displays an index of Help topics.

**General help**

Displays Help for the active window.

**Using help**

Describes how to use Help.

**How do I**

Displays task Help.

**VisualAge for C++ help menu choices**

Each choice launches its associated VisualAge for C++ documentation. These choices are available when you start the Performance Analyzer from WorkFrame.

**Product information**

Displays information about the Performance Analyzer.

**Pop-up  
Menus**

Pop-up menus allow you to quickly access features that are frequently used. The **Window Manager** window has the following pop-up menus:

- **Window Manager Executable** pop-up menu
- **Window Manager Trace File** pop-up menu
- **Window Manager Diagram** pop-up menu.

**Window Manager Executable Pop-up Menu:** To access this pop-up menu, click mouse button 2 on the file name of an executable or the icon next to it. The menu is displayed with the following choices:

**Create trace**

Displays the window that lets you start creating a trace file. The file name on which you clicked appears in the window's **Program Name** entry field.

**Close**

Closes the selected executable, its associated trace file, and all diagrams in which the trace data is displayed.

**Window Manager Trace File Pop-up Menu:** To access this pop-up menu, click mouse button 2 on the file name of a trace file or the icon next to it. The menu is displayed with the following choices:

**Analyze trace**

Displays the window from which you can select Performance Analyzer diagrams to analyze the trace file. The file name on which you clicked appears in the window's **Trace File Name** entry field.

**Open as Call Nesting**

Displays the trace file in the **Call Nesting** diagram.

**Open as Dynamic Call Graph**

Displays the trace file in the **Dynamic Call Graph**.

**Open as Execution Density**

Displays the trace file in the **Execution Density** diagram.

**Open as Statistics**

Displays the trace file in the **Statistics** diagram.

**Open as Time Line**

Displays the trace file in the **Time Line** diagram.

**Close**

Closes the selected trace file and all diagrams in which the trace file is displayed.

**Delete file**

Closes all diagrams in which the trace file is displayed, and then deletes the selected trace file from your hard disk if you select **Yes** in the **Trace File - Delete** window (displayed when you select this choice).

**Window Manager Diagram Pop-up Menu:** To access this pop-up menu, click mouse button 2 on the file name of a diagram or the icon next to it. The menu is displayed with the following choices:

**Display**

Makes the selected diagram active and brings it to the foreground of your desktop.

**Close**

Closes the selected diagram.

**Push Buttons** The **Window Manager** window has the following push buttons:

**Create Trace**

Displays the **Create Trace** window from which you can start creating a trace file.

**Analyze Trace**

Displays the **Analyze Trace** window from which you can select Performance Analyzer diagrams to analyze a trace file.

---

## Create Trace Window

The **Create Trace** window lets you specify the name of the program that you want to trace. Optional fields allow you to: enter parameters that you can pass to your program; specify a custom trace file name; and attach a description to the trace file.

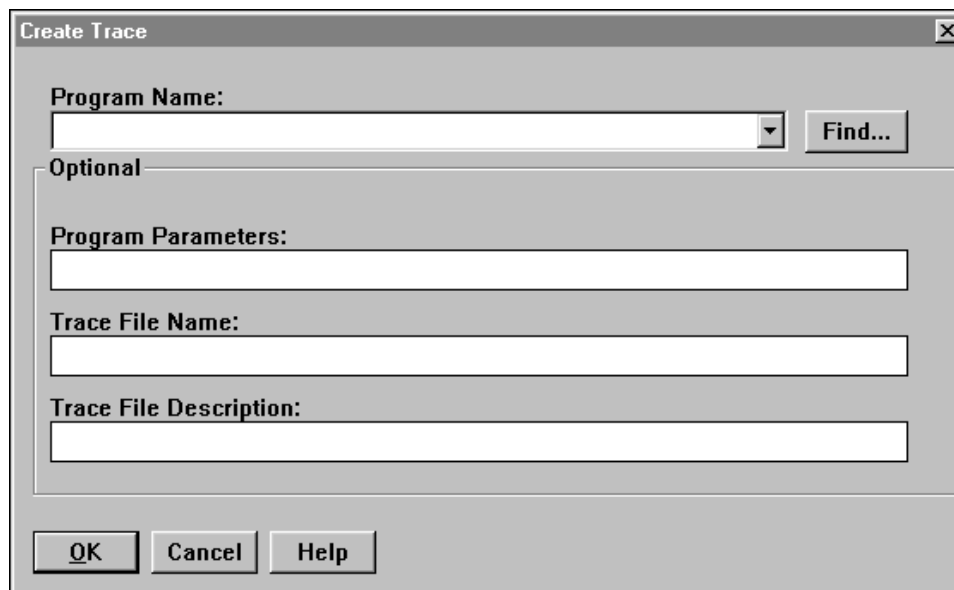


Figure 141. Create Trace Window

You can display the **Create Trace** window from the **Performance Analyzer - Window Manager** window by:

- Clicking on the **Create Trace** push button
- Selecting the **Create trace** choice from the **File** menu
- Clicking mouse button 2 on an executable file name or icon (if displayed in the window), and selecting the **Create trace** choice from the pop-up menu.

## Areas of the Create Trace Window

The following topics describe the areas of the **Create Trace** window.

<b>Program Name Entry Field</b>	Type the full path name and program you want to trace in the <b>Program Name</b> entry field. If the program is in your current directory, you do not have to type the path.  <b>Note:</b> If you are not sure where the file is located, select the <b>Find</b> push button.
<b>Optional Entry Fields</b>	The <b>Create Trace</b> window has the following optional entry fields:  <b>Program Parameters</b> Type any parameters that you want to pass to your program in the <b>Program Parameters</b> entry field.  <b>Trace File Name</b> Type a name for the trace file in the <b>Trace File Name</b> entry field if you do not want the default.  The Performance Analyzer places the trace file in the executable's directory unless you specify a path. The default name for the trace file is <i>myprog.trc</i> , where <i>myprog</i> is the name of the program you are tracing.  <b>Trace File Description</b> Type any comments that you want to make about your trace in the <b>Trace File Description</b> entry field.

**Push Buttons** The **Create Trace** window has the following push buttons:

<b>Find</b>	Displays the <b>Find File</b> window, which helps you locate a file that you want to trace.
<b>OK</b>	Saves the changes and closes the window.
<b>Cancel</b>	Exits the window without saving any changes.
<b>Help</b>	Displays information about the window.

**Note:** Before creating a trace file, you must compile and link your program with the proper options as described in “Compiling and Linking Your Program” on page 433.

---

## Trace Generation Window

The **Trace Generation** window lists the file names of the preloaded components in your program and lets you control which parts of your program are traced.

A component can be an executable (EXE) file, a dynamic link library (DLL) file, an object file, or a function. EXE and DLL files contain object files, and object files contain functions.

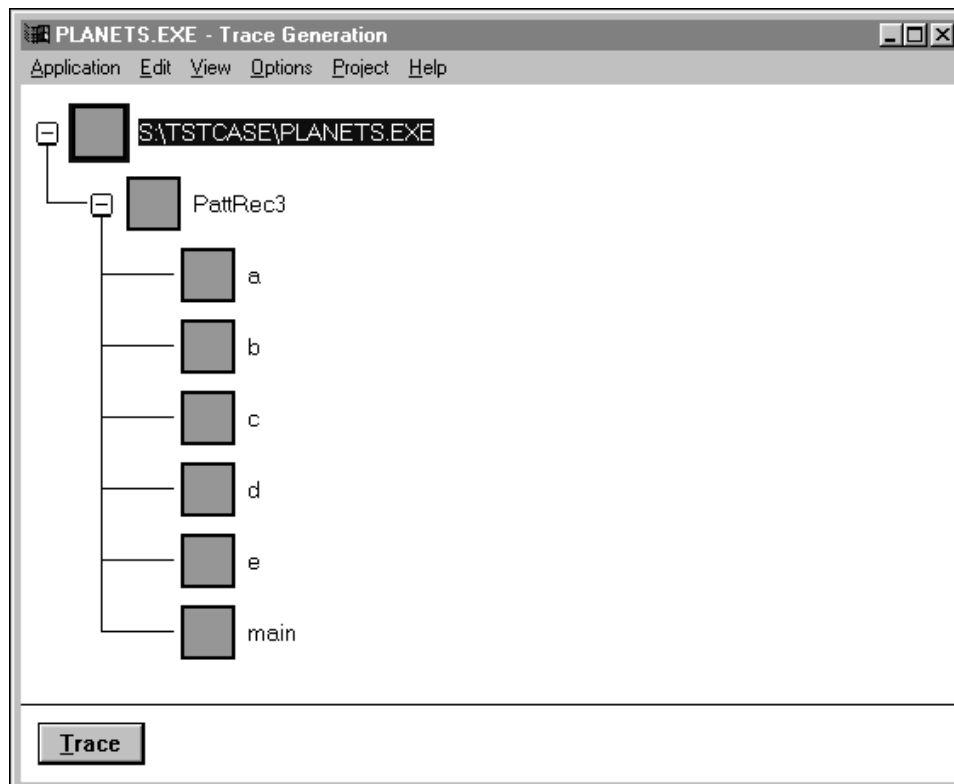


Figure 142. Trace Generation Window

To view or hide components, click on the plus or minus icon to expand or contract EXE, DLL, and object files.

**Note:** Before creating a trace file, you must compile and link your program with the proper options as described in “Compiling and Linking Your Program” on page 433.

## Areas of the Trace Generation Window

The following topics describe the areas of the **Trace Generation** window.

### Menu Bar Summary

The menu choices in the **Trace Generation** window are as follows:

#### Application

From this menu, you can select:

#### Window Manager

Displays the **Performance Analyzer - Window Manager** window.

#### Exit the Performance Analyzer

Lets you end the Performance Analyzer application.

## **Edit**

The **Edit** menu is a dynamic menu, which displays choices based on the type of component selected. From this menu, you can select:

### **Enable all executables**

Enables all functions in all executable files. (When a component is *enabled*, the Performance Analyzer records trace analysis data for it in the trace file.)

### **Disable all executables**

Disables all functions in all executable files. (When a component is *disabled*, the Performance Analyzer does not record trace analysis data for it in the trace file.)

### **Enable executable**

Enables all functions in a selected executable file. This choice is available when you select a disabled executable.

### **Disable executable**

Disables all functions in a selected executable file. This choice is available when you select an enabled executable.

### **Enable object file**

Enables all functions in a selected object file. This choice is available when you select a disabled object file.

### **Disable object file**

Disables all functions in a selected object file. This choice is available when you select an enabled object file.

### **Enable function**

Enables a function. This choice is available when you select a disabled function.


### **Disable function**

Disables a function. This choice is available when you select an enabled function.

### **Set trigger**

Sets a trigger on a function so that the Performance Analyzer traces the function and its associated calls. This choice is available when you select a function that *does not* have a trigger set on it.

**Note:** If triggers are set, the Performance Analyzer starts tracing when it reaches the triggered function and continues tracing until it receives a return from that function. If no triggers are set, the Performance Analyzer traces all enabled components.

	<p><b>Remove trigger</b></p> <p>Removes a trigger on a function. This choice is available when you select a function that <i>has</i> a trigger set on it.</p> <p><b>Note:</b> If triggers are set, the Performance Analyzer starts tracing when it reaches the triggered function and continues tracing until it receives a return from that function. If no triggers are set, the Performance Analyzer traces all enabled components.</p>
<b>View</b>	<p>From this menu, you can select:</p> <p><b>Traceable filter</b></p> <p>Displays only the components that are traceable. For a component to be traceable, it must have been properly compiled and linked, and it must be linked with CPPWPA3.OBJ.</p> <p> For instructions on compiling and linking your program, see “Compiling and Linking Your Program” on page 433.</p> <p><b>Tree lines</b></p> <p>Displays lines between components to show relationships.</p>
<b>Options</b>	<p>The choices on this menu let you set options to customize your trace sessions. From this menu, you can select:</p> <p><b>Buffer wrap</b></p> <p>Enables or disables buffer wrapping.</p> <p><b>Call depth</b></p> <p>Allows you to select the call depth limit for each thread.</p> <p><b>Font...</b></p> <p>Displays the <b>Font</b> window, which lets you change the font, font style, and font size for the <b>Trace Generation</b> window.</p> <p><b>Name trace file</b></p> <p>Allows you to specify a new path and/or file name for the trace file. Also allows you to add or change the trace file description.</p> <p><b>Timeout control</b></p> <p>Allows you to select the maximum number of seconds your program may run without logging events. This choice is useful when your program is in a continuous loop or deadlock.</p>



	<b>Time stamp events</b> Enables or disables time stamps.
	<b>Settings &gt;</b> Displays a cascaded menu that lets you select either of the following:
<b>Project</b>	This menu appears on the <b>Trace Generation</b> window when you start the Performance Analyzer from within the WorkFrame environment. From this menu, you can select:
	<b>Edit source</b> Displays the source file for a selected object file or function in WorkFrame's default editor.
	<b>WorkFrame actions</b> WorkFrame actions that can be launched from the <b>Trace Generation</b> window will also appear in this menu.
<b>Help</b>	This menu provides choices that display various types of Help information. From this menu, you can select:
	<b>Help index</b> Displays an index of Help topics.
	<b>General help</b> Displays Help for the active window. The online Help panel displayed is the same panel that is displayed when you place your cursor inside the window and press F1.
	<b>Using help</b> Describes how to use Help.
	<b>How do I</b> Displays task help.
	<b>VisualAge for C++ help menu choices</b> Each choice launches its associated VisualAge for C++ documentation. These choices are available when you start the Performance Analyzer from WorkFrame.
	<b>Product information</b> Displays information about the Performance Analyzer.

## Pop-up Menus

Pop-up menus allow you to quickly access features that are frequently used. The **Trace Generation** window has a pop-up menu for each type of program component displayed on the window. The pop-up menu displayed depends on the type of component you click on.

The **Trace Generation** window pop-up menus are as follows:

- **Trace Generation Executable** pop-up menu
- **Trace Generation Object File** pop-up menu
- **Trace Generation Function** pop-up menu.

**Trace Generation Executable Pop-up Menu:** To access this pop-up menu, click mouse button 2 on the file name of an executable or dynamic link library file (or the plus/minus icon next to it). If you click on the file name of a component other than an executable, the pop-up menu you see will be different.

The menu is displayed with the following choices:

### **Disable executable**

Disables the selected executable file so that the Performance Analyzer *does not* record trace analysis data for it in the trace file. This choice is available when you select an *enabled* executable.

### **Enable executable**

Enables the selected executable file so that the Performance Analyzer records trace analysis data for it in the trace file. This choice is available when you select a *disabled* executable.

**Trace Generation Object File Pop-up Menu:** To access this pop-up menu, click mouse button 2 on the file name of an object file (or the plus/minus icon next to it). If you click on the file name of a component other than an executable, the pop-up menu you see will be different.

The menu is displayed with the following choices:

**Disable object file**

Disables the selected object file so that the Performance Analyzer *does not* record trace analysis data for it in the trace file. This choice is available when you select an *enabled* object file.

**Enable object file**

Enables the selected object file so that the Performance Analyzer records trace analysis data for it in the trace file. This choice is available when you select a *disabled* object file.

**Edit source**

Displays the source file for a selected object file in WorkFrame's default editor. The Performance Analyzer finds the source file and opens it to the first line of the file.

**Trace Generation Function Pop-up Menu:** To access this pop-up menu, click mouse button 2 on the name of a function (or the plus/minus icon next to it). If you click on the file name of a component other than an executable, the pop-up menu you see will be different.

The menu is displayed with the following choices:

**Disable function**

Disables the selected function so that the Performance Analyzer *does not* record trace analysis data for it in the trace file. This choice is available when you select an *enabled* function.

**Enable function**

Enables the selected function so that the Performance Analyzer records trace analysis data for it in the trace file. This choice is available when you select a *disabled* function.

**Set trigger**

Sets a trigger on a function so that the Performance Analyzer traces the function and its associated calls. This choice is available when you select a function that *does not* have a trigger set on it.

**Note:** If triggers are set, the Performance Analyzer starts tracing when it reaches the triggered function and continues tracing until it receives a return from that function. If no triggers are set, the Performance Analyzer traces all enabled components.

**Remove trigger**

Removes a trigger on a function. This choice is available when you select a function that *has* a trigger set on it.

**Note:** If one or more triggers are set, the Performance Analyzer starts tracing when it reaches the triggered function and continues tracing until

it receives a return from that function. If no triggers are set, the Performance Analyzer traces all enabled components.

#### **Edit source**

Displays the source file for a selected function in WorkFrame's default editor. The Performance Analyzer finds the source file and opens it to the line where the function begins.

**Push Button** The **Trace Generation** window has the following push button:

**Trace** Use the **Trace** push button to close the **Trace Generation** window and begin tracing your program.

---

## **Application Monitor Window**

After you select the **Trace** push button on the **Trace Generation** window to start tracing your program, the Performance Analyzer displays the **Application Monitor** window.



Figure 143. Application Monitor Window

This window is displayed until your entire program has executed or you select the **Stop** push button, which causes your program to stop executing. When you stop the execution of your program, you also stop the collection of trace data.

### **Areas of the Application Monitor Window**

The following topics describe the areas of the **Application Monitor** window.

**Status Area** The following information is displayed in the **Application Monitor** window **Status Area**:

- Name of the program being traced
- Name of the trace file
- Number of bytes written to the trace file
- Number of events written to the trace file.

**Push Buttons** The **Application Monitor** window has the following push buttons:

**Stop** Stops the execution of your program and the collection of trace data.

The **Analyze Trace** window is displayed after the trace stops so that you can select one or more diagrams in which to view the trace data.

**Trace on** Causes tracing to begin. **Trace on** does not cause your program to start running; it causes trace events to start being recorded to the trace file. When the **Application Monitor** window is displayed, the Performance Analyzer has already started tracing events, so you cannot select **Trace on** until you have first selected **Trace off**.

You can also turn tracing on from your program by calling the Performance Analyzer function PerfStart. For instructions on calling PerfStart, see the Performance Analyzer online Help facility.

**Trace off** Causes tracing to stop. **Trace off** does not cause your program to stop running; it stops trace events from being recorded to the trace file. When the **Application Monitor** window is displayed, the Performance Analyzer has already started tracing events, so you can select **Trace off** anytime after the window is displayed.

**Note:** Return events are still logged for functions that were called before you selected Trace off, and execution time will still be logged against functions that were called but have not returned to the caller.

You can also turn tracing off from your program by calling the Performance Analyzer function PerfStop. For instructions on calling PerfStop, see the Performance Analyzer online Help facility.

---

## Analyze Trace Window

The **Analyze Trace** window lets you specify the name of the trace file that you want to analyze and the diagrams in which you want to display it.

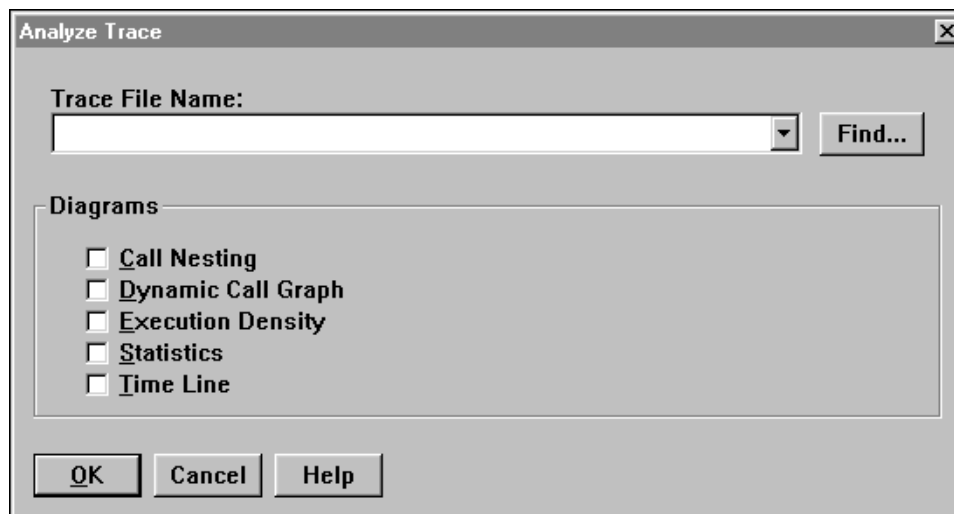


Figure 144. Analyze Trace Window

**Note:** You must have created a trace file before you can open a diagram. For instructions, see Chapter 34, “Creating a Trace File” on page 445.

You can display the **Analyze Trace** window from the **Performance Analyzer - Window Manager** window by:

- Clicking on the **Analyze Trace** push button
- Selecting the **Analyze trace** choice from the **File** menu
- Clicking mouse button 2 on a trace file name or icon (if displayed in the window), and selecting the **Analyze trace** choice from the pop-up menu.

## Areas of the Analyze Trace Window

The following topics describe the areas of the **Analyze Trace** window.

**Trace File Name Entry Field** Type the full path and file name of the trace file that you want to analyze in the **Trace File Name** entry field. If the file is in your current directory, you do not have to type the path.

**Note:** If you are not sure where the file is located, select the **Find** push button.

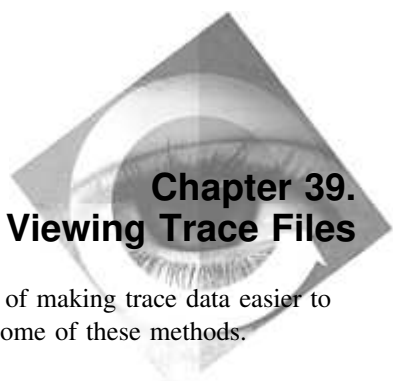
**Diagrams Check Boxes** Select one or more check boxes. The trace file will be displayed in the diagram or diagrams selected.

**Push Buttons** The **Analyze Trace** window has the following push buttons:

<b>OK</b>	Saves the changes and closes the window.
<b>Cancel</b>	Exits the window without saving any changes.
<b>Help</b>	Displays Help information for the window.







## Chapter 39. Viewing Trace Files

The Performance Analyzer diagrams provide methods of making trace data easier to view and understand. The following topics describe some of these methods.

---

### Using Filtering

Filters allow you to temporarily reduce the amount of trace data displayed in a diagram. There are several techniques for filtering the trace file and isolating interesting or problematic areas.

The following list contains filtering techniques that you can use in each of the diagrams:

#### Call Nesting

In this diagram, you can filter data by:

- Selecting specific functions and threads to view. The Performance Analyzer displays trace information for only those threads or functions selected.
- Selecting a region of time to view.

#### Dynamic Call Graph

In this diagram, you can filter data by:

- Selecting specific threads to view. The Performance Analyzer displays trace information for only those threads selected.
- Displaying data by function, class (if you are analyzing a C++ program), or executable. Using **View > Nodes of >**, you can select which component (function, class, or executable) the diagram's nodes represent.
- Scaling node sizes to change the size of the graphical representation of components.

#### Execution Density

In this diagram, you can filter data by:

- Selecting specific functions and threads to view. The Performance Analyzer displays trace information for only those selected.
- Scaling the diagram to reveal more information.
- Selecting a region of time to view.

#### Statistics

In this diagram, you can filter data by:

- Selecting specific threads to view. The Performance Analyzer displays trace information for only those threads selected.
- Displaying data by function, class (if you are analyzing a C++ program), or executable. Using **View > Details on >**, you can

select which type of component data (function, class, or executable) is displayed in the diagram's **Details** pane.

#### **Time Line**

In this diagram, you can filter data by:

- Scaling the diagram to reveal more information.
- Selecting a region of time to view.

---

## **Using Zooming**

The **Dynamic Call Graph**, **Execution Density**, and **Time Line** diagrams have zooming capabilities that allow you to enlarge (**Zoom in**) or reduce (**Zoom out**) the size of the diagram.

Use **Zoom in** to focus on regions of a diagram that are of most interest.

The **Overview** choice on the **Dynamic Call Graph** allows you to quickly move around in the diagram, and zoom in and out of the diagram. The gray box in the **Overview** window highlights the area currently displayed in the **Dynamic Call Graph**.

- Click and hold mouse button 1 inside the gray box, and then move it around in the **Overview** window to quickly change the view in the **Dynamic Call Graph**.
- Grab the sides of the gray box to resize the area shown in the **Dynamic Call Graph**.

---

## **Using Scaling**

Scaling is a way to change how much detail is displayed. You can view the diagram as a whole or magnify areas to see the finer detail.

When details are hidden, you cannot spot patterns, anomalies, or features of the execution because there is too much information on the screen. The **Execution Density** and **Time Line** diagrams can be scaled along the time dimension.

---

## **Using Scrolling**

You can scroll the window in all diagrams to focus on areas of interest. You can also use correlation to scroll to areas of interest in each of the chronologically-scaled diagrams (**Call Nesting**, **Execution Density**, and **Time Line**). Correlation is described in “Understanding Correlation” on page 477.

---

## Using Multiple Views

The Performance Analyzer allows you to open a trace file in several diagrams or multiple views of the same diagram simultaneously. Sometimes opening two or more diagrams can help you better understand a program.

For instance, if you have a new program to learn, and you don't want to wade through code listings to determine how the code works, you can display and use the **Dynamic Call Graph**, **Call Nesting**, and **Time Line** diagrams to get a good understanding of the program's flow.

After you have opened a diagram, you can quickly open another diagram by clicking on the appropriate icon in the tool bar, or by selecting the **Open as** choice from a **Trace file** menu, and then selecting another diagram from the cascaded menu.

---

## Recognizing Patterns

Program loops cause the same sequence of calls and returns to be repeated in the trace. The Performance Analyzer lets you combine like sequences in the **Call Nesting** diagram by selecting **Use pattern recognition**.

Pattern recognition looks at a single thread and finds patterns of calls and returns. When this choice is enabled, the **Call Nesting** diagram groups these patterns using curved arcs. The number of repetitions is shown to the right.

This technique shortens the number of pages which you must scroll through to look at your trace file.

If you see a pattern repeated numerous times, you could improve performance by grouping the functions in the pattern together with *pragma alloc\_text* statements. This will limit the number of page swaps between calls in the patterns.

**Note:** The **Use pattern recognition** check box (shown when you select **View > Include threads...**) is only selectable when you are filtering by threads.

---

## Understanding Correlation

Correlation allows you to mark a point in time in one diagram and then find that same point in another diagram.

Using correlation is helpful because one diagram cannot show everything of interest within a trace file. Additionally, some events are easier to find in one diagram, but the information in another is more meaningful; therefore, you can locate the event in one diagram and correlate to another.

The Performance Analyzer provides three time-scaled diagrams that can be correlated: **Call Nesting**, **Execution Density**, and **Time Line**. You can correlate these diagrams based on a specific time or event, or on a range of time or events.

For example, use the **Call Nesting** diagram to identify the order and names of functions called, and then use the **Time Line** diagram to find out how long a function took to execute.

Or you can use the **Execution Density** diagram to see general patterns that led up to a certain point, and then correlate that point to the **Call Nesting** diagram to see the exact order of the function calls.

For instructions on correlating diagrams, see the Performance Analyzer online Help facility.

## Chapter 40. Call Nesting Diagram

The **Call Nesting** diagram shows the trace file as a vertical series of function calls and returns. Use this diagram to diagnose problems with critical sections, sequencing protocols, program deadlocks, program crashes, and thread delays.

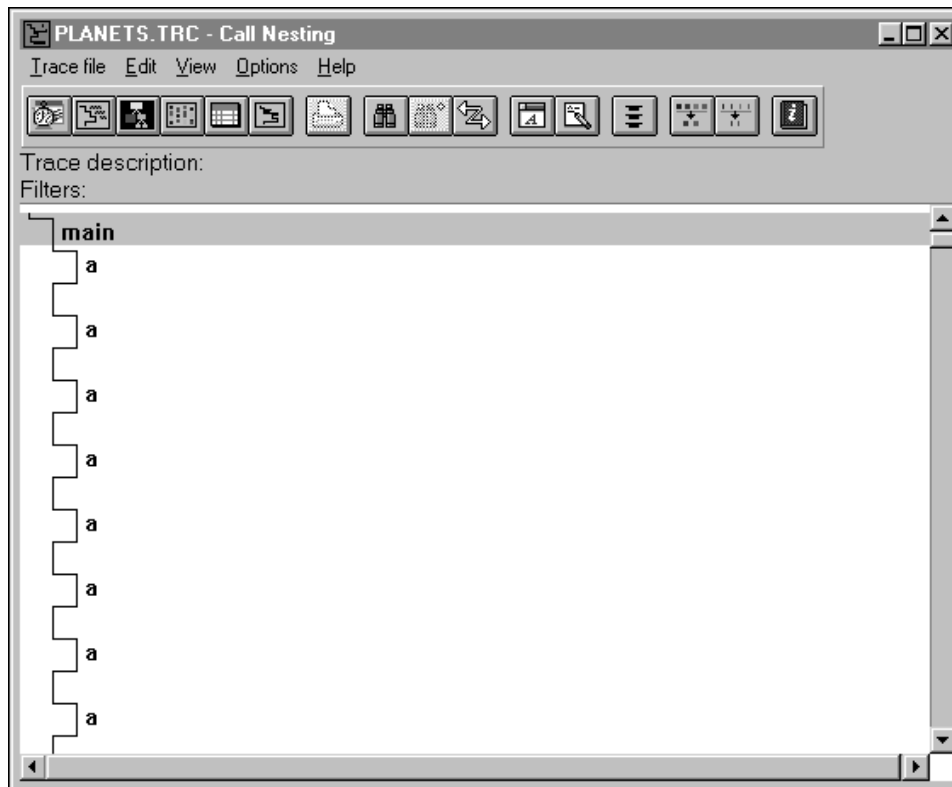


Figure 145. Call Nesting Diagram

A call is shown as a step to the right and a return is shown as a line back to the left. The calls are labeled with the name of the function being called.

Use the mouse to select a call, a return, or a user event. (User events and annotations are text strings in the diagram.) When the call is selected, it is highlighted.

Context switches between threads are shown by dashed horizontal lines. While the vertical lines do not show elapsed scaled times in this diagram, you can clearly see the flow of control and the interactions among the various threads.

---

## Areas of the Call Nesting Diagram

The following topics describe the areas of the **Call Nesting** diagram.

### Call Nesting Menu Bar Summary

The menu choices in the **Call Nesting** diagram are as follows:

**Trace file** Allows you to perform functions with your existing trace file. From this menu, you can select:

**Open as >**

Displays a cascaded menu with the following choices. Select the name of the diagram you want to view.

- **Call Nesting**
- **Dynamic Call Graph**
- **Execution Density**
- **Statistics**
- **Time Line**

**Printer settings...**

Allows you to select a printer and various options for your printed output.

**Print selected region...**

Prints a selected area of the diagram.

**Window Manager**

Displays the **Performance Analyzer - Window Manager** window.

**Exit the Performance Analyzer**

Lets you end the Performance Analyzer application.

**Edit** Allows you to locate and change text in the **Call Nesting** diagram. From this menu, you can select:

**Find >**

Displays a cascaded menu from which you can select:

- Function** Locates a function call or return.
- User event** Locates a user event.
- Annotation** Locates annotated text.

**Find next**

Locates the next occurrence of the last item you searched for.

**Annotate...**

Allows you to insert comments into your diagram.

	<b>Select time...</b> Allows you to go to a specific time in the diagram.
	<b>Select time range...</b> Allows you to select all events in a specified time range.
	<b>Select all</b> Selects the entire <b>Call Nesting</b> diagram.
<b>View</b>	Allows you to change displayed information. From this menu, you can select: <ul style="list-style-type: none"> <li> <b>Include functions...</b>            Controls which functions are included in or excluded from the diagram. Each selected function and any functions in its call chain will be shown.         </li> <li> <b>Include threads...</b>            Controls which threads are included in the diagram and lets you select pattern recognition.         </li> </ul>
<b>Options</b>	Allows you to customize the <b>Call Nesting</b> diagram and display additional information. From this menu, you can select: <ul style="list-style-type: none"> <li> <b>Call stack...</b>            Displays all the functions on the call stack at a selected point.         </li> <li> <b>Correlation...</b>            Synchronizes other diagrams to display the same highlighted region.         </li> <li> <b>Font...</b>            Displays the <b>Font</b> window, which lets you select the font, font style, and font size for the function names.         </li> <li> <b>Thread layout...</b>            Lets you select the amount of indentation for each thread column, and choose whether separator bars are drawn between threads.         </li> <li> <b>Status area...</b>            Allows you to select which items appear in the area at the top of the window.         </li> </ul>

**Tool bar >**

Displays a cascaded menu that lets you select:

**Show**

Lets you choose to either show or hide the tool bar.

**Hover help**

Lets you choose to either enable or disable displaying the descriptive text when the mouse pointer hovers over the tool bar buttons.

**Settings >**

Displays a cascaded menu that lets you select:

**Save**

Saves the current session settings.

**Restore initial defaults**

Restores the original settings.

**Project** This menu appears when you start the Performance Analyzer from within the WorkFrame environment. From this menu, you can select:

**Edit function**

Displays the source code for the selected function in the default editor for WorkFrame's edit action.

WorkFrame actions that can be launched from the **Call Nesting** diagram will also appear in this menu.



- Help** This menu provides choices that display various types of Help information. From this menu, you can select:
  - Help index** Displays an index of Help topics.
  - General help** Displays Help for the active window. The online Help panel displayed is the same panel that is displayed when you place your cursor inside the window and press F1.
  - Using help** Describes how to use Help.
  - How do I** Displays task help.
  - VisualAge for C++ help menu choices** Each choice launches its associated VisualAge for C++ documentation. These choices are available when you start the Performance Analyzer from WorkFrame.
  - Product information** Displays information about the Performance Analyzer.

## Call Nesting Status Area

The **Status Area**, located at the top of the window, describes the settings of the diagram. You can use the **Status area** choice on the **Options** menu to change what appears in the **Status Area**. When you select **Status area**, you can select whether you want the following to appear in the **Status Area** window:

- Trace description** Shows the trace file description (if one was entered when the trace file was created).
- Filters** Displays selected filters.

## Call Nesting Pop-up Menus

The pop-up menus allow you to quickly access features that are frequently used. The **Call Nesting** diagram has two pop-up menus: the **Call Nesting Diagram** pop-up menu and the **Call Nesting Selected Item** pop-up menu.

The **Call Nesting Diagram** pop-up menu contains actions that can be applied to the entire diagram, and the **Call Nesting Selected Item** pop-up menu contains actions that can be applied to a highlighted area.

**Pop-Up Menu** To access this pop-up menu, click mouse button 2 on the background area of the diagram. The menu is displayed with the following choices:

**Find >**

Displays a cascaded menu from which you can select:

**Function** Locates a function call or return.

**User event** Locates a user event.

**Annotation** Locates annotated text.

**Find next**

Locates the next occurrence of the last item you searched for.

**Include functions...**

Controls which functions are included in or excluded from the diagram.

Each selected function and any functions in its call chain will be shown.

**Include threads...**

Controls which threads are included in the diagram and lets you select pattern recognition.

**Font...**

Displays the **Font** window, which lets you select the font, font style, and font size for the function names.

**Thread layout...**

Lets you select the amount of indentation for each thread column, and choose whether separator bars are drawn between threads.

**Selected Item Pop-Up Menu** To access this pop-up menu, click mouse button 2 on the highlighted area of the diagram. The menu is displayed with the following choices:

**Annotate...**

Allows you to insert comments into your diagram.

**Call stack...**

Displays all the functions on the call stack at a selected point.

**Correlation...**

Synchronizes other diagrams to display the same highlighted region.

**Edit function**

Displays the source code for the selected function in the default editor for WorkFrame's edit action.

WorkFrame actions that can be launched from the **Call Nesting** diagram will also appear in this menu.

## Chapter 41. Dynamic Call Graph

The **Dynamic Call Graph** is a graphical representation of your program's execution. Use this diagram to:

- Get an overall view of your program and its flow.
- See the relative importance (in terms of execution time) of program components.
- See where time is spent in your program.
- See your program's call hierarchy.

**Note:** The Dynamic Call Graph is not an inheritance diagram.

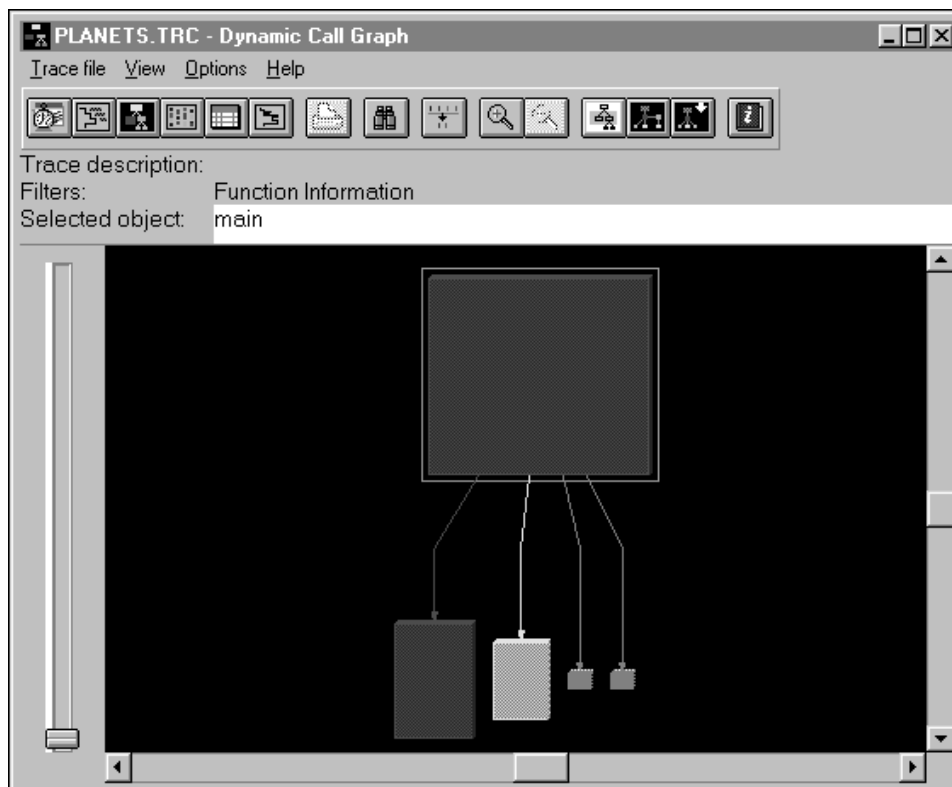



Figure 146. Dynamic Call Graph

In this diagram, nodes represent components. Components can be functions, classes (only if you are analyzing a C++ program), or executables.

Arcs, which are displayed between pairs of nodes, represent calls from one component to another. Only calls made during the given execution of the program are displayed.

The diagram's color code helps you quickly determine where time is spent in your program and estimate the relative number of calls that occur between functions, classes, or executables.

Additionally, the diagram uses node size to show you information about execution time and time on stack.

 For more information about the diagram's use of color and node size, see “Dynamic Call Graph Nodes and Arcs” on page 487.

To choose which type of component the nodes represent, select **View > Nodes of >**, and then select one of the following choices:

- **Functions.**
- **Classes.** (You can only select this choice if you are analyzing a C++ program.)
- **Executables.**

By default, the **Nodes of** setting is **Functions** for C programs and **Classes** for C++ programs.

Using the **Nodes of** choice to filter the diagram by function, class, or executable allows you to show the amount of detail that is meaningful to you. For example, you may want to view classes instead of functions in order to discern patterns or find problem areas more easily.

The **Status Area** displays the trace file description (if one was entered when the trace file was created), the type of component information displayed (function, class, or executable), and information about the selected node or arc.

---

## Dynamic Call Graph Nodes and Arcs

The **Dynamic Call Graph** uses nodes to represent program components (functions, classes [if you are analyzing a C++ program], or executables) and arcs to represent calls between components.

Color and node size have special significance in this diagram. Node color depicts the time spent in the node. Arc color depicts the number of calls between nodes. Node size for a function is based on the function's execution time and time on stack as they compare to the function or functions with the greatest execution time and time on stack. Node size for an executable or a class is based on execution time.

Selected arcs are green, and a green box surrounds a node when it is selected.

## Displaying Information about Nodes and Arcs

To display:

- The name of the component a node represents, click on the node to select it. (When selected, a node is surrounded by a green box.)  
The name of the component represented appears in the **Status Area**.
- Detailed information about the component a node represents, do either of the following:
  - Double-click on the node.
  - Click mouse button 2 on the node, and then select **Display information...** from the pop-up menu.

The window displayed using either of these methods identifies the component represented and reports other useful statistics, including execution time, number of calls, and time on stack. For classes, member function information is also displayed.

- The names of the components involved in a call, click on the arc representing the call to select it. (When selected, an arc is green.)  
The names of the components represented (the caller and callee) appear in the **Status Area**.
- Detailed information about the components involved in a call, do either of the following:
  - Double-click on the arc.
  - Click mouse button 2 on the arc, and then select **Display information...** from the pop-up menu.

The window displayed using either of these methods identifies the caller, the callee, the number of times the caller called the callee, and the total number of times the callee was called during the program execution.

## Node and Arc Color

Node color is based on the maximum execution time spent in any component. Arc color is based on the maximum number of calls between any two pairs of components.

The following table explains how the **Dynamic Call Graph** uses node color.

### Notes:

1. *Maximum execution time* is the amount of time required to execute the component that has the greatest execution time.
2. Class execution time is calculated by totalling the execution times for all member functions contained in the class.

Node Color:	Indicates:
Gray	Less than 1/8 of the maximum execution time was spent in the component.
Blue	1/8 to 1/4 of the maximum execution time was spent in the component.
Yellow	1/4 to 1/2 of the maximum execution time was spent in the component.
Red	Greater than 1/2 of the maximum execution time was spent in the component.

The following table explains how the **Dynamic Call Graph** uses arc color.

**Note:** *Maximum number of calls made* is the number of calls made between the two components that contained the greatest number of calls between them.

Arc Color:	Indicates:
Green	An arc is selected.
Gray	Less than 1/8 of the maximum number of calls made were made between components.
Blue	1/8 to 1/4 of the maximum number of calls made were made between components.
Yellow	1/4 to 1/2 of the maximum number of calls made were made between components.
Red	At least 1/2 of the maximum number of calls made were made between components.

## Node Size

For *functions*, node height is determined by the node's execution time relative to the function that has the greatest execution time.

Node width is determined by the node's time on stack relative to the function that has the greatest time on stack.

Time on stack is the time spent on the call stack. If a function is waiting for a message, the time spent waiting is logged as time on stack.

**Note:** Time on stack includes all time spent in subroutines.

For *executables* and *classes*, node height and width are based on execution time; therefore, the nodes representing these components will always be square.

Time on stack is not relevant for executables and classes.

---

## Areas of the Dynamic Call Graph

The following topics describe the areas of the **Dynamic Call Graph**.

### Dynamic Call Graph Menu Bar Summary

The menu choices in the **Dynamic Call Graph** are as follows:

**Trace file** Allows you to perform functions with your existing trace file. From this menu, you can select:

**Open as >**

Displays a cascaded menu with the following choices. Select the name of the diagram you want to view.

- **Call Nesting**
- **Dynamic Call Graph**
- **Execution Density**
- **Statistics**
- **Time Line**

**Printer settings...**

Allows you to select a printer and various options for your printed output.

**Print selected region...**

Prints a selected area of the diagram.

**Window Manager**

Displays the **Performance Analyzer - Window Manager** window.

**Exit the Performance Analyzer**

Lets you end the Performance Analyzer application.

**View** Allows you to change displayed information. From this menu, you can select:

**Nodes of >**

Lets you specify which component the diagram's nodes represent. From the cascaded menu, you can select:

**Functions** Specifies that the diagram's nodes represent functions.

**Classes** Specifies that the diagram's nodes represent classes. (You can only select this if you are analyzing a C++ program.)

**Executables** Specifies that the diagram's nodes represent executables.

**Include threads...**

Controls which threads are included in the diagram.

**Show arcs**

Displays or hides all arcs in the diagram.

**Overview**

Allows you to navigate through large diagrams quickly.

**Zoom in**

Magnifies the diagram to view a region of interest.

**Zoom out**

Reduces the scale of the diagram incrementally.

**Re-lay graph**

Sizes to fit all of the diagram in the window.

**Restore graph**

Restores nodes and arcs to the diagram to show the default view.

**Options** Allows you to customize the **Dynamic Call Graph** and display additional information. From this menu, you can select:

**Scale node sizes**

Scales node sizes.

**Find...**

Searches for functions, classes (only if you are analyzing a C++ program), or executables in the diagram.



**Status area...**

Allows you to select which items appear in the area at the top of the window.

**Tool bar >**

Displays a cascaded menu that lets you select:

**Show**

Lets you choose to either show or hide the tool bar.

**Hover help**

Lets you choose to either enable or disable displaying the help text when the mouse pointer hovers over the tool bar buttons.

**Settings >**

Displays a cascaded menu that lets you select either of the following:

**Save**

Saves the current session settings.

**Restore initial defaults**

Restores the original settings.

**Project**

This menu appears when you start the Performance Analyzer from within the WorkFrame environment. From this menu, you can select:

**Edit function**

Displays the source code for the selected function in the default editor for WorkFrame's edit action.

WorkFrame actions that can be launched from the **Dynamic Call Graph** will also appear in this menu.

**Help**

This menu provides choices that display various types of Help information. From this menu, you can select:

**Help index**

Displays an index of Help topics.

**General help**

Displays Help for the active window. The online Help panel displayed is the same panel that is displayed when you place your cursor inside the window and press F1.

**Using help**

Describes how to use Help.

**How do I**

Displays task Help.

### **VisualAge for C++ help menu choices**

Each choice launches its associated VisualAge for C++ documentation. These choices are available when you start the Performance Analyzer from WorkFrame.

### **Product information**

Displays information about the Performance Analyzer.

## **Dynamic Call Graph Status Area**

The **Status Area**, located at the top of the window, describes the settings of the diagram. You can select the **Status area** choice from the **Options** menu to change what appears in the **Status Area**. When you select **Status area**, you can select the following in the **Status Area** window:

### **Trace description**

Shows the trace file description (if one was entered when the trace file was created).

### **Filters**

Displays selected filters.

### **Selected object**

Select to display selected object.

**Note:** If you highlight a node, the component name is displayed. If you highlight an arc, the names of the components involved in the call are displayed.

## **Dynamic Call Graph Pop-up Menus**

The pop-up menus allow you to quickly access features that are frequently used. The **Dynamic Call Graph** has two pop-up menus: the **Dynamic Call Graph Selected Node** pop-up menu and the **Dynamic Call Graph Selected Arc** pop-up menu.

The **Dynamic Call Graph Selected Node** pop-up menu contains actions that can be applied to nodes, and the **Call Nesting Selected Arc** pop-up menu contains actions that can be applied to arcs.

**Selected Node Pop-Up Menu** To access this pop-up menu, click mouse button 2 on a node in the **Dynamic Call Graph**. The menu is displayed with the following choices:

- Display information...**  
Displays a window with information about the selected node.
- Center**  
Centers the selected node in the diagram.
- Hide node**  
Removes the selected node and all arcs connected to it from the diagram.
- Edit function**  
If you are working in the WorkFrame environment, this choice displays the source code for the selected function in the default editor for WorkFrame's edit action.

**Selected Arc Pop-Up Menu** To access this pop-up menu, click mouse button 2 on an arc in the **Dynamic Call Graph**. The menu is displayed with the following choices:

- Display information...**  
Displays a window with information about the selected arc.
- Center**  
Centers the selected arc in the diagram.

**Function Information Window** Use the **Function Information** window to display information about a selected function.

When the **Nodes of** choice is set to **Functions**, you can display the **Function Information** window by either double-clicking on a node or clicking mouse button 2 on a node, and then selecting **Display information...** from the pop-up menu.

The following information is provided in the **Function Information** window:

- Function name
- Object file name
- Executable name
- Execution time
- Number of calls
- Time on stack.

The following push buttons are also in the window:

**Who calls me**

Select this push button to display only the selected node, and the node or nodes that called the selected node. To return to the diagram, select **Restore graph** from the **View** menu.

**Whom do I call**

Select this push button to display only the selected node and the nodes that the selected node calls. To return to the diagram, select **Restore graph** from the **View** menu.

**Cancel**

Exits the window without saving any changes.

**Help**

Displays Help.

**Note:** If the trace file does not have time stamps:

- The *Execution Time* and *Time on stack* will not be displayed.
- All nodes will be displayed in the same color and size, and the **Scale node sizes** choice will be disabled.

**Class Information Window**

Use the **Class Information** window to display information about a selected class.

When the **Nodes of** choice is set to **Classes**, you can display the **Class Information** window by either double-clicking on a node or clicking mouse button 2 on a node, and then selecting **Display information...** from the pop-up menu.

The following information is provided in the **Class Information** window:

- Class name
- Names of member functions and their associated statistics
- Execution time
- Number of calls.

The following push buttons are also in the window:

**Who calls me**

Select this push button to display only the selected node, and the node or nodes that called the selected node. To return to the diagram, select **Restore graph** from the **View** menu.

**Whom do I call**

Select this push button to display only the selected node and the nodes that the selected node calls. To return to the diagram, select **Restore graph** from the **View** menu.

**Cancel**

Exits the window without saving any changes.

**Help**

Displays Help.

**Note:** If the trace file does not have time stamps:

- **Execution time** will not be displayed.

- All nodes will be displayed in the same color and size, and the **Scale node sizes** choice will be disabled.

### Executable Information Window

Use the **Executable Information** window to display information about a selected executable.

When the **Nodes of** choice is set to **Executables**, you can display the **Executable Information** window by either double-clicking on a node or clicking mouse button 2 on a node, and then selecting **Display information...** from the pop-up menu.

The following information is provided in the **Executable Information** window:

- Executable name
- Member functions for the executable and associated data
- Execution time
- Number of calls.

The following push buttons are also in the window:

#### Who calls me

Select this push button to display only the selected node, and the node or nodes that called the selected node. To return to the diagram, select **Restore graph** from the **View** menu.

#### Whom do I call

Select this push button to display only the selected node and the nodes that the selected node calls. To return to the diagram, select **Restore graph** from the **View** menu.

#### Cancel

Exits the window without saving any changes.

#### Help

Displays Help.

**Note:** If the trace file does not have time stamps:

- **Execution time** will not be displayed.
- All nodes will be displayed in the same color and size, and the **Scale node sizes** choice will be disabled.

## **Who Calls Whom Window**

Use the **Who Calls Whom** window to display information about a selected arc.

To display the **Who Calls Whom** window, either double-click on an arc or click mouse button 2 on an arc, and then select **Display information...** from the pop-up menu.

The **Who Calls Whom** window shows the name of the calling component (function, class, or executable), the called component, and the number of times the call was made.

The **Who Calls Whom** window has the following push buttons:

### **Find caller**

Selects and centers the node that made the call represented by the selected arc.

### **Find callee**

Selects and centers the node that is called by the call represented in the selected arc.

### **Cancel**

Exits the window without saving any changes.

### **Help**

Displays Help for the window.

## **Dynamic Call Graph Zoom Bar**

Move the slider on the **Zoom bar** up or down to quickly change the scale of the diagram view. When zooming in and out, the **Dynamic Call Graph** takes the selected node or arc (the node or arc that is highlighted) and uses it as the focal point.

The following list contains more ways to zoom in the **Dynamic Call Graph** diagram.

You can use the:

- **Zoom in** choice from the **View** menu to change the diagram view.  
When zooming in, your view is a step closer to the selected object and the object in the window appears larger.
- **Zoom out** choice from the **View** menu to change the diagram view.  
When zooming out, your view is a step further away from the selected object and the object in the window appears smaller.
- **Re-lay** choice from the **View** menu to return to the original diagram view.
- **Overview** choice from the **View** menu to quickly move around in the viewing area of the diagram.

When using the **Overview** choice, you have a miniature version of the **Dynamic Call Graph** window. The gray box highlights the area currently in view in the window.

Moving the small gray box around in the **Overview** window allows you to change the view in the **Dynamic Call Graph** quickly. You can also grab the sides of the gray box to resize the area being shown in the diagram window.





## Chapter 42. Execution Density Diagram

The **Execution Density** diagram shows trends of program execution by displaying the trace data chronologically from top to bottom as thin horizontal lines of various colors in different columns.

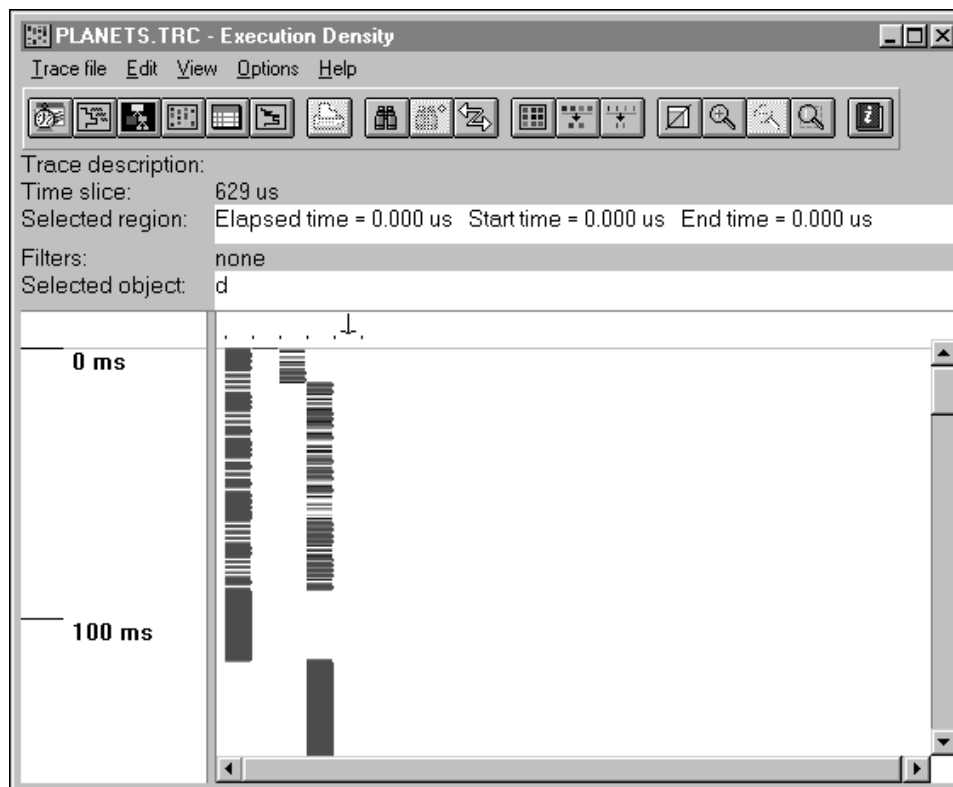


Figure 147. Execution Density Diagram

The following list describes the diagram's components:

- Each vertical column represents a function.
- The thickness of each line represents a unit of time called a time slice.
- The color of each line represents the percentage of program execution time spent in the given function for that time slice. Only selected threads are used in calculating this percentage.

For instance, in the default setting, functions executing more than 50% of a given time slice have a red line drawn in the appropriate column at the vertical location corresponding to the time slice.

**Note:** When a thread switch occurs in the **Execution Density** diagram, the time between the last recorded event in the previous thread and the first event in the new thread will be allotted to the previous thread.

Magnification is initially turned off, meaning the entire trace file is visible in the window. You can magnify or filter the diagram to vary the amount of detail displayed.

**Note:** Events that take small amounts of time might be hard to distinguish until the magnification has been increased.

---

## Areas of the Execution Density Diagram

The following topics describe the areas of the **Execution Density** diagram.

### Execution Density Menu Bar Summary

The menu choices in the **Execution Density** diagram are as follows:

**Trace file** Allows you to perform functions with your existing trace file. From this menu, you can select:

**Open as >**

Displays a cascaded menu with the following choices. Select the name of the diagram you want to view.

- **Call Nesting**
- **Dynamic Call Graph**
- **Execution Density**
- **Statistics**
- **Time Line**

**Printer settings...**

Allows you to select a printer and various options for your printed output.

**Print selected region...**

Prints a selected area of the diagram.

**Window Manager**

Displays the **Performance Analyzer - Window Manager** window.

**Exit the Performance Analyzer**

Lets you end the Performance Analyzer application.

<b>Edit</b>	Allows you to locate and change text in the <b>Execution Density</b> diagram. From this menu, you can select:
	<b>Find function...</b> Searches for a function.
	<b>Find next</b> Locates the next occurrence of the last item you searched for.
	<b>Select time...</b> Allows you to go to a specific time in the diagram.
	<b>Select time range...</b> Allows you to select all events in a specified time range.
	<b>Select all</b> Selects the entire <b>Execution Density</b> diagram.
<b>View</b>	Allows you to change displayed information. From this menu, you can select:
	<b>Zoom in</b> Magnifies the diagram to view a region of interest.
	<b>Zoom out</b> Reduces the scale of the diagram incrementally.
	<b>Zoom to selected range</b> Magnifies an area of interest.
	<b>Scale pages...</b> Allows you to zoom in on a particular point in time or zoom out to get an overview of the diagram.
	<b>Include functions...</b> Isolates specific functions to view in your program.
	<b>Include threads...</b> Controls which threads are included in the diagram.
<b>Options</b>	Allows you to customize the <b>Execution Density</b> diagram and display additional information. From this menu, you can select:
	<b>Color...</b> Selects the colors used to display the percentage of time used by each function and the percentage of time each color represents.
	<b>Column width...</b> Controls the width (in pixels) for each function column in the diagram.

**Correlation...**

Synchronizes other diagrams to display the same highlighted region.

**Status area...**

Allows you to select which items appear in the area at the top of the window.

**Tool bar >**

Displays a cascaded menu that lets you select:

**Show**

Lets you choose to either show or hide the tool bar.

**Hover help**

Lets you choose to either enable or disable displaying the help text when the mouse pointer hovers over the tool bar buttons.

**Settings >**

Displays a cascaded menu that lets you select either of the following:

**Save**

Saves the current session settings.

**Restore initial defaults**

Restores the original settings.

**Project** This menu appears when you start the Performance Analyzer from within the WorkFrame environment. From this menu, you can select:

**Edit function**

Displays the source code for the selected function or object file in the default editor for WorkFrame's edit action.

WorkFrame actions that can be launched from the **Execution Density** diagram will also appear in this menu.

**Help** This menu provides choices that display various types of Help information. From this menu, you can select:

**Help index**

Displays an index of Help topics.

**General help**

Displays Help for the active window.

**Using help**

Describes how to use Help.

**How do I**

Displays task Help.

### **VisualAge for C++ help menu choices**

Each choice launches its associated VisualAge for C++ documentation. These choices are available when you start the Performance Analyzer from WorkFrame.

### **Product information**

Displays information about the Performance Analyzer.

## **Execution Density Status Area**

The **Status Area**, located at the top of the window, describes the settings of the diagram. You can select the **Status area** choice from the **Options** menu to change what appears in the **Status Area**. When you select **Status area**, you can select the following in the **Status Area** window:

### **Trace description**

Shows the trace file description (if one was entered when the trace file was created).

### **Time slice**

Displays the value of the time slice.

### **Selected region**

Displays the total time, start time, and end time for a selected region.

### **Filters**

Displays selected filters.

### **Selected object**

Displays the name of the selected object.

## **Execution Density Pop-up Menus**

The pop-up menus allow you to quickly access features that are frequently used. The **Execution Density** diagram has two pop-up menus: the **Execution Density Diagram** pop-up menu and the **Execution Density Selected Item** pop-up menu.

**Pop-up Menu** This pop-up menu contains most of the choices from the **Edit** and **Options** menus. To access this pop-up menu, click mouse button 2 on the background area of the diagram. The menu is displayed with the following choices:

### **Find function...**

Searches for a function.

### **Find next**

Locates the next occurrence of the last item you searched for.

### **Zoom in**

Magnifies the diagram to view a region of interest.

### **Zoom out**

Reduces the scale of the diagram incrementally.

**Scale pages...**

Allows you to zoom in on a particular point in time or zoom out to get an overview of the diagram.

**Include functions...**

Isolates specific functions to view in your program.

**Include threads...**

Controls which threads are included in the diagram.

**Color...**

Selects the colors used to display the percentage of time used by each function and the percentage of time each color represents.

**Column width...**

Controls the width (in pixels) for each function column in the diagram.

**Selected Item Pop-up Menu** To access this pop-up menu, highlight a region of interest, move the mouse pointer into that area, and click on mouse button 2. This menu is displayed with the following choices:

**Zoom to selected range**

Magnifies an area of interest.

**Correlation...**

Synchronizes other diagrams to display the same highlighted region.

**Edit function**

Displays the source code for the selected function or object file in the default editor for WorkFrame's edit action.

WorkFrame actions that can be launched from the **Execution Density** diagram will also appear in this menu.

---

## Execution Density Current Column Indicator

An arrow called the current column indicator is displayed at the top of the columns. The current column indicator lets you use the mouse to select each column.

To move the current column indicator, click on the column of interest and the arrow moves. You can drag the current column indicator arrow with the mouse and the **Selected object** information will change.

---

## Execution Density Vertical Ruler

The **Vertical Ruler**, located to the left of the diagram, shows the approximate time of events on the screen. To change the time scale, select the **Scale pages** choice from the **View** menu, and the **Scale Pages** window appears.

## Chapter 43. Statistics Diagram

The **Statistics** diagram gives you a textual report of execution time by function, class (only if you are analyzing a C++ program), or executable. Use this information to find hot spots in the overall program execution.

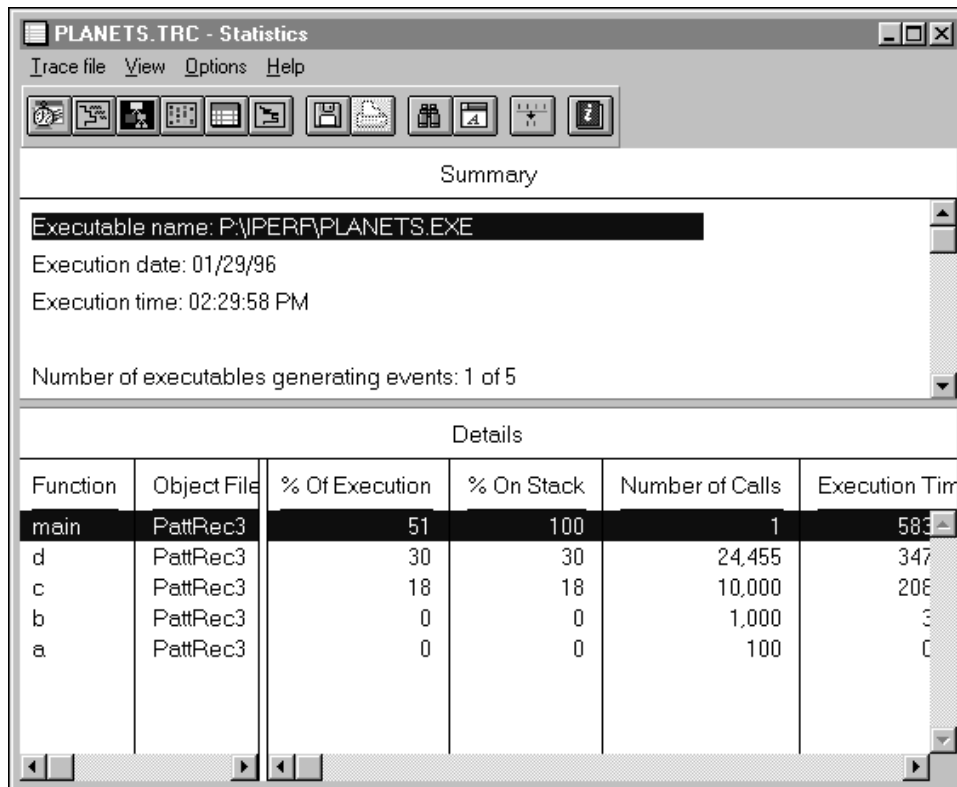


Figure 148. Statistics Diagram

This diagram has two panes: **Summary** and **Details**.

The **Summary** pane lists general facts about your program's overall execution.

The **Details** pane lists specific facts about the execution of individual program components. You can select the type of information you want displayed in this pane. If you want to display information about:

- Functions, select **View > Details on > Functions**.

- Classes, select **View > Details on > Classes**.

**Note:** You can only select this choice if you are analyzing a C++ program.

- Executables, select **View > Details on > Executables**.

You can resize the panes vertically. To resize a pane, position the mouse pointer on the split bar, and then press and hold mouse button 1 as you drag the bar. Release the mouse button when the pane is the size you want.

**Note:** Times are shown in milliseconds.

---

## Areas of the Statistics Diagram

The following topics describe the areas of the **Statistics** diagram.

### Statistics Menu Bar Summary

The menu choices in the **Statistics** diagram are as follows:

**Trace file** Allows you to perform functions with your existing trace file. From this menu, you can select:

**Open as >**

Displays a cascaded menu with the following choices. Select the name of the diagram you want to view.

- **Call Nesting**
- **Dynamic Call Graph**
- **Execution Density**
- **Statistics**
- **Time Line**

**Save as text...**

Creates a file containing the statistical information.

**Printer settings...**

Allows you to select a printer and various options for your printed output.

**Print...**

Prints the textual report of the **Statistics** diagram.

**Window Manager**

Displays the **Performance Analyzer - Window Manager** window.

**Exit the Performance Analyzer**

Lets you end the Performance Analyzer application.



<b>View</b>	Allows you to change displayed information. From this menu, you can select:						
	<b>Details on &gt;</b> Allows you to specify how the <b>Details</b> window will be displayed. A cascaded menu contains the following choices: <table> <tr> <td><b>Functions</b></td><td>Specifies that the <b>Details</b> window display function information.</td></tr> <tr> <td><b>Classes</b></td><td>Specifies that the <b>Details</b> window display class information. (Only selectable if you are analyzing a C++ program.)</td></tr> <tr> <td><b>Executables</b></td><td>Specifies that the <b>Details</b> window display executable information.</td></tr> </table>	<b>Functions</b>	Specifies that the <b>Details</b> window display function information.	<b>Classes</b>	Specifies that the <b>Details</b> window display class information. (Only selectable if you are analyzing a C++ program.)	<b>Executables</b>	Specifies that the <b>Details</b> window display executable information.
<b>Functions</b>	Specifies that the <b>Details</b> window display function information.						
<b>Classes</b>	Specifies that the <b>Details</b> window display class information. (Only selectable if you are analyzing a C++ program.)						
<b>Executables</b>	Specifies that the <b>Details</b> window display executable information.						
	<b>Include threads...</b> Controls which threads are included in the diagram.						
	<b>Sort &gt;</b> Select how you want to sort the <b>Details</b> window. A cascaded menu contains the following choices. Invalid choices are grayed out. <ul style="list-style-type: none"> <li>• <b>Function name</b></li> <li>• <b>Class name</b></li> <li>• <b>Object file name</b></li> <li>• <b>Executable name</b></li> <li>• <b>Execution time</b></li> <li>• <b>Time on call stack</b></li> <li>• <b>Number of calls</b></li> <li>• <b>Minimum call</b></li> <li>• <b>Maximum call</b></li> <li>• <b>Average call</b></li> </ul>						
<b>Options</b>	Allows you to customize the <b>Statistics</b> diagram and display additional information. From this menu, you can select:						
	<b>Find...</b> Searches for functions, classes (only if you are analyzing a C++ program), or executables.						
	<b>Font...</b> Displays the <b>Font</b> window, which lets you select the font and font size for the <b>Summary</b> and <b>Details</b> panes.						
	<b>Tool bar &gt;</b> Displays a cascaded menu that lets you select:						

**Show**

Lets you choose to either show or hide the tool bar.

**Hover help**

Lets you choose to either enable or disable displaying the help text when the mouse pointer hovers over the tool bar buttons.

**Settings >**

Displays a cascaded menu that lets you select either of the following:

**Save**

Saves the current session settings.

**Restore initial defaults**

Restores the original settings.

**Project** This menu appears when you start the Performance Analyzer from within the WorkFrame environment. From this menu, you can select:

**Edit function**

Displays the source code for the selected function in the default editor for WorkFrame's edit action.

WorkFrame actions that can be launched from the **Statistics** diagram will also appear in this menu.

**Help** This menu provides choices that display various types of Help information. From this menu, you can select:

**Help index**

Displays an index of Help topics.

**General help**

Displays Help for the active window.

**Using help**

Describes how to use Help.

**How do I**

Displays task Help.

**VisualAge for C++ help menu choices**

Each choice launches its associated VisualAge for C++ documentation. These choices are available when you start the Performance Analyzer from WorkFrame.

**Product information**

Displays information about the Performance Analyzer.

## Statistics Summary Pane

The **Summary** pane is in the top area of the window. You must scroll the pane to view all of the information.

To resize the pane vertically, position the mouse pointer on the split bar, and then press and hold mouse button 1 as you drag the bar. Release the mouse button when the pane is the size you want.

Information provided in the **Summary** pane is as follows:

- Executable name
- Trace file description

**Note:** This will only appear if a description was entered when the file was created.

- Execution date
- Execution time
- Number of executables generating events
- Number of classes generating events (only if you are analyzing a C++ program)
- Number of functions generating events
- Number of threads generating events
- Total number of events
- Total number of annotations
- Number of user events
- Maximum call nest depth
- Number of trace buffer flushes
- Total trace time excluding overhead
- Trace overhead

## Statistics Details Pane

The **Details** pane is in the bottom area of the window. You must scroll the pane to view all of the information.

To resize the pane vertically, position the mouse pointer on the split bar, and then press and hold mouse button 1 as you drag the bar. Release the mouse button when the pane is the size you want.

The **Details** pane has a left and right pane. The left pane displays the fully-qualified name of the component on which the statistics have been gathered.

Other information provided on the left side of the **Details** pane varies depending on the **View > Details on >** setting.

If **Details on** is set to:

- **Functions**, you see columns with the following headings: *Function*, *Object file*, and *Executable*.
- **Classes**, you see one column with the heading *Class*.
- **Executables**, you see one column with the heading *Executable*.

With function names, if user events have been included in the trace, they will appear as separate entries in the list of function names.

User events will list the function name that made the call to the user event, followed by the user event in parentheses.

For executables, only the executable name is displayed.

Information provided on the right side of the **Details** pane varies depending on which type of component is selected in **View > Details on >**. This information could include the following:

- % Of Execution
- % On Stack
- Number of Calls
- Execution Time
- Time on Stack
- Minimum Call
- Maximum Call
- Average Call.

**Note:** If you disabled the **Time stamp events** choice before you created your trace file, only the *Number of Calls* column will be displayed on the right side of the **Details** pane.

## Chapter 44. Time Line Diagram

The **Time Line** diagram displays the sequence of nested function calls and returns. Time stamps determine the exact placement of an event along the time dimension on the vertical axis. This provides a direct and natural presentation of the chronological relationships of events.

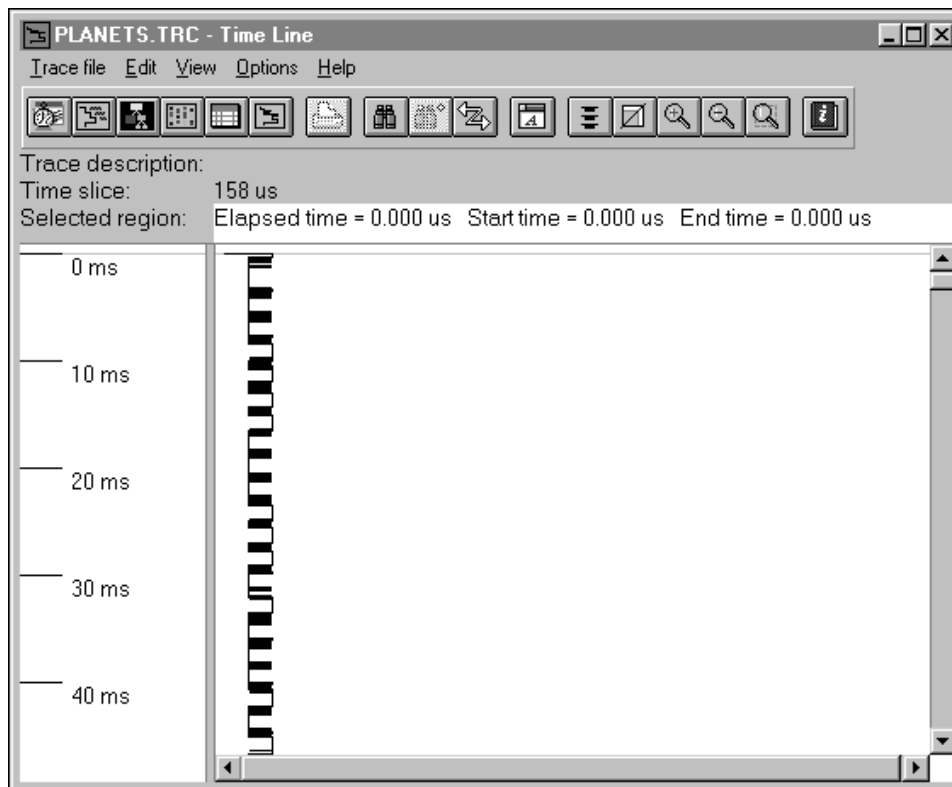


Figure 149. Time Line Diagram

The **Time Line** diagram is similar to the **Call Nesting** diagram, but the distance between successive events in the diagram are drawn in proportion to the actual time between the events as recorded in the trace file.

The names of functions are only drawn when the time spent in that function is large enough to allow the name to be drawn. This value is dependent upon the size of the font being used. For example, with the default font, the distance between the call to

the function and the next event must be at least twenty scan lines. This is done to ensure that the function name will not be overwritten by another function name or overdrawn by a line representing a function call or thread switch.

You can use this diagram to find where a deadlock occurred. Access violations, system exceptions, and other such program errors are recorded in the trace file as *user events*, as are any messages generated in the code by calls to the Performance Analyzer. These events are indicated by a pink circle in the diagram, and if there is sufficient space, the text associated will be drawn to the right of the events.

---

## Areas of the Time Line Diagram

The following topics describe the areas of the **Time Line** diagram.

### Time Line Menu Bar Summary

The menu choices in the **Time Line** diagram are as follows:

**Trace file** Allows you to perform functions with your existing trace file. From this menu, you can select:

**Open as >**

Displays a cascaded menu with the following choices. Select the name of the diagram you want to view.

- **Call Nesting**
- **Dynamic Call Graph**
- **Execution Density**
- **Statistics**
- **Time Line**

**Printer settings...**

Allows you to select a printer and various options for your printed output.

**Print selected region...**

Prints a selected area of the diagram.

**Window Manager**

Displays the **Performance Analyzer - Window Manager** window.

**Exit the Performance Analyzer**

Lets you end the Performance Analyzer application.

**Edit** Allows you to locate and change text in the **Time Line** diagram. From this menu, you can select:

<b>Find</b>	<b>Find &gt;</b>	Displays a cascaded menu from which you can select:
	<b>Function</b>	Locates a function call or return.
	<b>User event</b>	Locates a user event.
	<b>Find next</b>	Locates the next occurrence of the last item you searched for.
	<b>Select time...</b>	Allows you to go to a specific time in the diagram.
	<b>Select time range...</b>	Allows you to select all events in a specified time range.
	<b>Select all</b>	Selects the entire <b>Time Line</b> diagram.
	<b>View</b>	Allows you to change displayed information. From this menu, you can select:
	<b>Zoom in</b>	Magnifies the diagram to view a region of interest.
	<b>Zoom out</b>	Reduces the scale of the diagram incrementally.
	<b>Zoom to selected range</b>	Magnifies the diagram to focus on the highlighted area.
	<b>Scale pages...</b>	Allows you to zoom in on a particular point in time or zoom out to get an overview of the diagram.
	<b>Options</b>	Allows you to customize the <b>Time Line</b> diagram and display additional information. From this menu, you can select:
	<b>Call stack...</b>	Displays all the functions on the call stack at a selected point.
	<b>Correlation...</b>	Synchronizes other diagrams to display the same highlighted region.
	<b>Font...</b>	Displays the <b>Font</b> window, which lets you select the font, font style, and font size for the diagram.
	<b>Thread layout...</b>	Lets you select the amount of indentation for each thread column, and choose whether separator bars are drawn between threads.

	<b>Status area...</b> Allows you to select which items appear in the area at the top of the window.
	<b>Tool bar &gt;</b> Displays a cascaded menu that lets you select: <ul style="list-style-type: none"> <li><b>Show</b> Lets you choose to either show or hide the tool bar.</li> <li><b>Hover help</b> Lets you choose to either enable or disable displaying the help text when the mouse pointer hovers over the tool bar buttons.</li> </ul>
	<b>Settings &gt;</b> Displays a cascaded menu that lets you select either of the following: <ul style="list-style-type: none"> <li><b>Save</b> Saves the current session settings.</li> <li><b>Restore initial defaults</b> Restores the original settings.</li> </ul>
<b>Project</b>	This menu appears when you start the Performance Analyzer from within the WorkFrame environment. From this menu, you can select: <ul style="list-style-type: none"> <li><b>Edit function</b> Displays the source code for the selected function in the default editor for WorkFrame's edit action.</li> </ul> WorkFrame actions that can be launched from the <b>Time Line</b> diagram will also appear in this menu.
<b>Help</b>	This menu provides choices that display various types of Help information. From this menu, you can select: <ul style="list-style-type: none"> <li><b>Help index</b> Displays an index of Help topics.</li> <li><b>General help</b> Displays Help for the active window.</li> <li><b>Using help</b> Describes how to use Help.</li> <li><b>How do I</b> Displays task Help.</li> </ul>



### **VisualAge for C++ help menu choices**

Each choice launches its associated VisualAge for C++ documentation. These choices are available when you start the Performance Analyzer from WorkFrame.

### **Product information**

Displays information about the Performance Analyzer.

## **Time Line Status Area**

The **Status Area**, located at the top of the window, describes the settings of the diagram. You can select the **Status area** choice from the **Options** menu to change what appears in the **Status Area**. When you select **Status area**, you can select the following in the **Status Area** window:

### **Trace description**

Shows the trace file description (if one was entered when the trace file was created).

### **Time slice**

Displays the value of the time slice.

### **Selected region**

Displays the total time, start time, and end time for a selected region.

## **Time Line Pop-up Menus**

The pop-up menus allow you to quickly access features that are frequently used. The **Time Line** diagram has two pop-up menus: the **Time Line Diagram** pop-up menu and the **Time Line Selected Item** pop-up menu.

**Pop-up Menu** This pop-up menu contains almost all of the choices from the **Edit** and **Options** menus. To access this pop-up menu, click mouse button 2 on the background area of the diagram. The menu is displayed with the following choices:

### **Find function...**

Searches for a function call or return.

### **Find next**

Locates the next occurrence of the last item you searched for.

### **Zoom in**

Magnifies the diagram to view a region of interest.

### **Zoom out**

Reduces the scale of the diagram incrementally.

### **Scale pages...**

Allows you to zoom in on a particular point in time or zoom out to get an overview of the diagram.

**Font...**

Displays the **Font** window, which lets you select the font, font style, and font size for the diagram.

**Thread layout...**

Lets you select the amount of indentation for each thread column, and choose whether separator bars are drawn between threads.

**Selected Item Pop-up Menu** To access this pop-up menu, highlight a region of interest, move the mouse pointer into that area, and click on mouse button 2. The menu is displayed with the following choices:

**Zoom to selected range**

Magnifies the diagram to focus on the highlighted area.

**Call stack...**

Displays all the functions on the call stack at a selected point.

**Correlation...**

Synchronizes other diagrams to display the same highlighted region.

**Edit function**

Displays the source code for the selected function in the default editor for WorkFrame's edit action.

WorkFrame actions that can be launched from the **Time Line** diagram will also appear in this menu.

---

## Time Line Vertical Ruler

The **Vertical Ruler**, located to the left of the diagram, shows the approximate time of events on the screen. To change the time scale, select the **Scale pages** choice from the **View** menu, and the **Scale Pages** window appears.

---

## Part 8. Managing Libraries

This part of the *User's Guide* describes tools that can help you manage libraries.

To create and maintain library files (.lib) and module definition files (.def), use the IBM Library Manager (ILIB) utility.

If you are using DLL files, you can use the DLLRNAME utility to globally rename a DLL.

---

<b>Chapter 45. Using IBM Library Manager</b> . . . . .	519
Running ILIB . . . . .	519
Controlling ILIB Input . . . . .	523
Controlling ILIB Output . . . . .	523
ILIB Objects . . . . .	525
ILIB Options . . . . .	528
 <b>Chapter 46. Module Definition Files</b> . . . . .	 533
Reserved Words . . . . .	534
Summary of Module Statements . . . . .	536
Module Statements . . . . .	536
 <b>Chapter 47. Packaging the VisualAge for C++ Runtime DLLs</b> . . . . .	 543
Using the DLLRNAME Utility . . . . .	544
How DLLRNAME Works . . . . .	545
DLLRNAME Options . . . . .	546
An Example . . . . .	547

---





## Chapter 45. Using IBM Library Manager

Use the IBM Library Manager (also referred to as ILIB in this reference) to create and maintain libraries of object code, create import libraries and export object pairs, and generate module definition (.def) files. Using the ILIB utility, you can:

- Create a new library from a collection of objects
- Maintain a library
  - Add objects to an existing library
  - Delete objects from an existing library
  - Copy objects from an existing library
  - Replace objects in an existing library
- List the contents of a new or existing library
- Create import library/export object pairs from:
  - Module definition (.def) files
  - Objects generated from source files containing `#pragma export` and `_Export` statements
  - A combination of the above
- Generate module definition (.def) files from:
  - An existing DLL
  - Objects generated from source files containing `#pragma export` and `_Export` statements
  - A combination of the above

---

### Running ILIB

Run ILIB by typing `ilib` at the command prompt.

You can specify parameters in the following ways:

1. Enter them directly on the command line.
2. Use the ILIB environment variable.
3. Put them in a text file, called a response file, and specify the file name after the `ilib` command.
4. A combination of the above

You can press `Ctrl+C` or `Ctrl+Break` at any time while running ILIB to return to the operating system. Interrupting ILIB before completion restores the original library from a backup.

## Using ILIB

### Notes:


1. When started, ILIB makes a backup copy of the original library in case it is interrupted or a mistake is made. Make sure you have enough disk space for both your original library and the modified copy.
2. The library must end with the extension `.lib`. If an extension is not specified, the default extension, `.lib`, will be appended. High Performance File System (HPFS) file names are supported. Hence, `mylibraryname.new.lib` is still a valid library.


## Using the Command Line

You can specify all the input ILIB needs on the command line. The syntax of the command line is:

```
ilib [options] [libraries] [@responsefile] [objects]
```

<b>options</b>	Options that affect the behavior of ILIB
<b>libraries</b>	The input library to be created or modified
<b>responsefile</b>	The name of a text file containing ILIB options
<b>objects</b>	Commands used to add, delete, replace, copy, and move object modules within the library

The ILIB command line is a free format command line; that is, the input arguments can be specified any number of times, in any order. The only exception is the `/FREEFORMAT` option, which does have a position restriction.  See “`/FREEFORMAT`, `/NOFREEFORMAT`” on page 530 for more information.

**Note:** For compatibility with the OS/2 release of ILIB, a fixed format command line is also supported. To use the fixed format command line, the `/NOFREEFORMAT` option must be specified immediately following `ilib` on the command line, or as the first parameter in the ILIB environment variable ( see “Using the ILIB Environment Variable” for more information). The default command line format is free format.

For the purposes of this document, only the free format command line will be described in detail. If you need more information on the fixed format command line, please refer to the *IBM VisualAge C++ for OS/2 User's Guide, V3.0*.

## Using the ILIB Environment Variable

You can use the ILIB environment variable to specify any default ILIB options. When the `ilib` command is invoked, the environment variable will be parsed before the command line.

Use the `SET` command to give value to the ILIB environment variable. You can do this in the following ways:

## Using ILIB

### Command Line

When the SET command is used on the command line, the values you specify are in effect for only that session. They override values previously specified.

You can append the original value of the variable using *%variable%*. For example,

```
SET ILIB=/FREEFORMAT %ILIB%
```

would cause the ILIB environment variable to be set to the original value of the ILIB environment variable, with the /NOFREEFORMAT option specified ahead of any existing options.

### Windows NT Control Panel

Windows NT allows you to update environment variables and have them take effect immediately (i.e. no reboot required) using the Windows NT Control Panel.

To set the ILIB environment variable:

- Select the **Main** group by double-clicking on the **Main** icon.
- Select the **System** icon from the **Main** group by double-clicking on it.
- Enter ILIB in the **Variable** field.
- Enter the value for the ILIB environment variable in the **Value** field.
- Choose **Set**.

### Windows 95 AUTOEXEC.BAT file

Windows 95 allows you to set environment variables in the AUTOEXEC.BAT file. Any environment variables set in this fashion are available in every user session.

Add a line to your AUTOEXEC.BAT file that sets the environment variable to the value you want. For example,

```
SET ILIB=/NOBROWSE /NOBACKUP
```

Because environment variables specified in your AUTOEXEC.BAT file are in effect for every session you start, this is a good place to specify options that you want to apply each time you invoke ILIB. However, after you make a change to your AUTOEXEC.BAT file, you must reboot your system to have the change take effect.

## Using an ILIB Response File

To provide input to ILIB with a response file, type:

```
ilib @responsefile
```

where *responsefile* is the name of a file containing the same information that can be specified on the command line.

### Why use a response file?

Use a response file for:

## Using ILIB

- complex and long commands you type frequently
- strings of commands that exceed the limit for command line length

A response file extends the command line to include everything in the response file. To split input to ILIB between the command line and a response file, put part of your input on the command line and specify a response file (preceding the response file name with the at sign (@)). No space can appear between the at sign and the file name.

The response file name can be any valid Windows file name. To use special characters in the file name, such as a space or the @ symbol, the file name must be enclosed in quotes.

ILIB responds to input you place in a response file just as it does to input you enter on a command line. Any newline characters that occur between arguments are treated as spaces. This allows you to extend an ILIB command to multiple lines.

**Note:** The options which specify which format command line to use (/FREEFORMAT or /NOFREEFORMAT) must be specified as the first parameter following `ilib` on the command line or as the first parameter in the ILIB environment variable. They cannot be specified inside the response file.

## Specifying ILIB Parameters - Examples



The following examples show different methods for specifying parameters to ILIB.

The operations shown in each example create a new library, `newlib.lib`, and its listing file, `newlib.lst`, from the existing `mylib.lib` library. `mylib.lib` is unchanged, but `newlib.lib` has these changes:

- the module text is deleted
- the object file `root.obj` is appended as an object module with the name `root`
- the module table is deleted and is replaced by a new table which is appended after `root`
- the module `string` is copied into an object file named `string.obj`

### Command Line Method

At the command line prompt, enter the following:

```
ilib /out:newlib.lib /list:newlib.lst mylib.lib /remove:text root table /extract:string
```

### Response File Method

First, create a response file with the following contents:



## Using ILIB

```
/out:newlib.lib  
/list:newlib.lst  
mylib.lib  
/remove:text  
root table  
/extract:string
```

Then, assuming the name of the response file is `response.fil`, invoke ILIB with:

```
ilib @response.fil
```

---

### Controlling ILIB Input

ILIB determines the format of any input files by examining the file contents. Most file formats can be identified by the file header information. If the format of an input file is not recognized and seems to contain only ASCII, it is assumed to be a module definition (`.def`) file.

ILIB allows you to place any extension you choose on a file and still have it dealt with correctly.

---

### Controlling ILIB Output

ILIB determines what output is to be produced by examining the options that you supply on the command line. The following options control ILIB output:

<i>Option</i>	<i>Description</i>
<code>/O[UT]:filename</code>	A static library is produced.
<code>/GEND[EF]:filename</code>	A module definition ( <code>.def</code> ) file is produced. The short form, <b>/gd</b> , may also be used.
<code>/GENI[MPLIB]:filename</code>	An import library/export object pair is produced. The short form, <b>/gi</b> , may also be used.
<code>/L[IST]:filename</code>	A list file is produced.

If none of the above are specified, ILIB will determine what is to be produced, as follows:


- If a DEF file is input to ILIB, an import library/export object pair will be produced.

**Note:** If there are no exported symbols, then no import library will be produced.

- If a library and/or object(s) are input to ILIB, a library combining them will be produced.

## Using ILIB

ILIB will allow you to generate a DEF file directly from a DLL. However, since the only information that a DLL has in it is the undecorated (exported) names, symbol decoration (calling convention) and type information (function or data) cannot be determined. ILIB will assume that all symbols exported from the DLL are `_Optlink` (the default linkage convention), unless an object file is provided that indicates otherwise.

The best way of using ILIB with a DLL is to use ILIB to create a DEF file using the `/gd` option. Edit the DEF file to change decorations, where appropriate, and then run the DEF file through ILIB using the `/gi` option to produce an import library/export object pair. For more information on creating and maintaining DEF files,  see Chapter 46, “Module Definition Files” on page 533.

### Notes:

1. If an import library/export object pair is requested, and only a DLL is specified as input, ILIB will generate an error.
2. The ILINK requires that an export (EXP) file be provided, along with object files, to build a DLL. If you use ICC to link your DLL and supply only the object files and a DEF file as input, ICC will detect the absence of an EXP file and will automatically invoke ILIB with the object files and DEF file to build the EXP file. Then ICC will invoke the ILINK with the object files and EXP file to build the DLL.

## Controlling ILIB Output - Examples



The following are examples showing how to control ILIB output.

### Library

The invocation

```
ilib /out:newlib.lib text.obj mylib.lib
```

will create the library `newlib.lib` out of the objects in `text.obj` and `mylib.lib`.

**Note:** Unless `newlib.lib` is specified as an input file, its contents will not be included in the library. If an output file already exists, and is not used as an input file, it will be replaced.

### DEF File

The invocation

```
ilib /gd:winner.def winner.dll
```

will create the module definition file `winner.def` from the DLL `winner.dll`.

**Import Library/Export Object Pair**

The invocation

```
ilib /gi winner.def
```

will create an import library named `winner.lib` and an export object named `winner.exp`. However, if no exported symbols are contained in `winner.def`, then `winner.lib` will not be produced.

**List File**

The invocation

```
ilib /list:mylib.lst mylib.lib
```

will generate the list file `mylib.lst`, based on the library `mylib.lib`, in the current directory.

---

**ILIB Objects**

ILIB objects are used to manipulate modules in a library. When you run ILIB, you can specify multiple objects in any order.

Each object consists of the ILIB command, followed by the name of the object module that is the subject of the command. Separate objects on the command line with a space or tab character.

**Summary of ILIB Objects**

The following is a summary of ILIB objects.

*Figure 150. ILIB Objects*

Syntax	Description	Default	Page
<i>filename</i>	Add/replace the named object in the library	None	526
/E[XTRACT]: <i>obj</i>	Copy the named object into the current directory and overwrite it if it already exists	None	527
/R[EMOVE]: <i>obj</i>	Remove the named object from the list of objects to be placed in the output library	None	527

## Using ILIB

### Notes:

1. ILIB objects are not case sensitive, so you can specify them in lower-, upper-, or mixed-case.  
  
You can also substitute a dash (-) for the slash (/) preceding the object. For example, -REMOVE:*filename* is equivalent to /REMOVE:*filename*.
2. You can specify objects in either short or long form. For example, /R:*filename* and /RE:*filename* are equivalent to /REMOVE:*filename*. See the table above for the shortest acceptable form of each object.
3. The order of operations when processing the command line is left to right.
4. ILIB never makes changes to your input library while it runs. It copies the library and makes changes to the copy. If ILIB is interrupted, your original library will be restored.

If you do not specify an output library, ILIB will not produce any output.

### Add/Replace Object

#### Syntax:

*filename*

#### Default:

Add *filename*

The default action, when *filename* is specified on the command line without an associated object, is to add it to the library. If *filename* already exists in the library, it will be replaced.

### Adding an Object Module to a Library

Type the name of the object file to be added on the command line. The .obj extension may be omitted.

ILIB uses the base name of the object file as the name of the object module in the library. For example, if the object file `cursor.obj` is added to a library file, the name of the corresponding object module is `cursor`.

Object modules are always added to the end of a library file.

### Replacing an Object Module in a Library

Type the name of the object module to be replaced on the command line. The .obj extension may be omitted.

If the object module already exists in the library, ILIB will replace it with the new copy.

## Using ILIB

### Combining Two Libraries

Specify the name of the library file to be added, including the `.lib` extension, on the command line. A copy of the contents of that library is added to the library file being modified. If both libraries contain a module with the same name, ILIB generates a warning message, and uses only the first module with that name.

ILIB adds the modules of the library to the end of the library being changed. Note that the added library still exists as an independent library because ILIB copies the modules without deleting them.



#### Examples

```
ilib /out:mylib.lib mylib.lib sample.obj
```

The command above adds the file `sample.obj` to the library `mylib.lib`. If `sample.obj` already exists in the library `mylib.lib`, ILIB will replace it.

```
ilib /out:newlib.lib newlib.lib mylib.lib
```

The command above adds the contents of the library `mylib.lib` to the library `newlib.lib`. The library `mylib.lib` is unchanged after this command is executed.

### /EXTRACT

#### Syntax:

`/E[EXTRACT]:obj`

#### Default:

None

Use `/EXTRACT` to copy a module from the library into an object file of the same name. The module remains in the library.

When ILIB copies the module to an object file, it adds the `.obj` extension to the module name and places the file in the current directory. If a file with this name already exists, ILIB overwrites it.



#### Example

```
ilib mylib.lib /extract:sample
```

The command above copies the module `sample` from the `mylib.lib` library to a file called `sample.obj` in the current directory. The module `sample` in `mylib.lib` is not altered.

### /REMOVE

#### Syntax:

`/R[REMOVE]:obj`

#### Default:

None

## Using ILIB

Use /REMOVE to delete an object module from a library. After /REMOVE, specify the name of the module to be deleted. Module names do not have path names or extensions.



### Examples

```
ilib /out:mylib.lib mylib.lib /remove:sample
```


The command above deletes the module sample from the library mylib.lib.

```
ilib /out:mylib.lib mylib.lib /extract:sample /remove:sample
```

The command above copies sample.obj from the mylib.lib library to an object file in the current directory. Then sample.obj is deleted from the library.

---

## ILIB Options

ILIB options affect the behavior of ILIB. When you run ILIB, you can specify multiple options in any order. The only exception is the /FREEFORMAT option, which has a position restriction.  See “/FREEFORMAT, /NOFREEFORMAT” on page 530 for details.

Separate options on the command line with a space or tab character.

## Summary of ILIB Options

The following is a summary of ILIB options.

Figure 151 (Page 1 of 2). ILIB Options

Syntax	Description	Default	Page
/?	Display help	None	529
/BA[CKUP] /NOBA[CKUP]	Back up the output file (if it exists) before overwriting it	/BA	529
/BR[OWSE] /NOBR[OWSE]	Include browse information in an OMF library	/BR	530
/DEF:def	Specify the name of a .def file to use to get information about exported symbols and linker parameters	None	530
/F[REEFORMAT] /NOF[REEFORMAT]	Use the free format command line	/F	530
/GEND[EF]:filename	Generate a .def file	None	530
/GENI[MPLIB]:filename	Generate an import library/export object pair	None	531
/H[ELP]	Display help	None	531
/L[IST]:filename	Generate a list file	None	531

## Using ILIB

Figure 151 (Page 2 of 2). ILIB Options

Syntax	Description	Default	Page
/NOE[XTDICTIONARY] /EXTD[ictionary]	Do not generate an extended dictionary in an OMF library	/EXTD	531
/O[UT]: <i>filename</i>	Specify the name of the output library	None	532
/Q[UIET], /NOL[OGO] /LO[GO], /NOQ[UIET]	Do not display the banner on startup	/LO	532
/W[ARN: <i>msgnum,msgnum</i> [,...]] /NOW[ARN: <i>msgnum,msgnum</i> [,...]]	Enable printing of warning message number <i>msgnum</i>	None	532

### Notes:

1. ILIB options are not case sensitive, so you can specify them in lower-, upper-, or mixed-case.  
  
You can also substitute a dash (-) for the slash (/) preceding the option. For example, -FREEFORMAT is equivalent to /FREEFORMAT.
2. You can specify options in either short or long form. For example, /F, /FR, and /FREE are equivalent to /FREEFORMAT. See the table above for the shortest acceptable form of each object.

See below for detailed information on each ILIB option.

### /?

#### Syntax:

/?

#### Default:

None

Use /? to display a list of valid ILIB options. This option is equivalent to /HELP.

### /BACKUP, /NOBACKUP

#### Syntax:

/BA[CKUP]  
/NOBA[CKUP]

#### Default:

/BACKUP

Use /BACKUP to back up the output file (if it exists) before overwriting it.

ILIB uses the base name of the library as the name of the backup library, and then appends the .bak extension. For example, if the library being modified is mylib.lib and a backup is requested, ILIB will create mylib.bak in the current directory.

## Using ILIB

### /BROWSE, /NOBROWSE

**Syntax:**  
/BR[OWSE]  
/NOBR[OWSE]

**Default:**  
/BROWSE


Use /NOBROWSE to exclude browse information from the output library. The VisualAge for C++ Browser can browse libraries and executable files that contain browse information. If you exclude browse information from the library, it cannot be browsed.

### /DEF

**Syntax:**  
/DEF[:filename]

**Default:**  
None

Use /DEF to specify the name of the .def file to use to get information about exported symbols and linker parameters.

This option is not required, since ILIB will recognize .def files by their contents if they are placed with other input files on the command line (see  “Controlling ILIB Input” on page 523 ).

### /FREEFORMAT, /NOFREEFORMAT

**Syntax:**  
/F[REEFORMAT]  
/NOF[REEFORMAT]

**Default:**  
/FREEFORMAT

Use the /FREEFORMAT option to tell ILIB that you are using the free format command line. The free format command line allows you to specify ILIB input arguments any number of times, in any order.

**Note:** This option must be specified immediately following `ilib` on the command line, or as the first argument in the ILIB environment variable. If you don't specify either /FREEFORMAT or /NOFREEFORMAT, ILIB will default to the free format command line.

### /GENDEF

**Syntax:**  
/GEND[EF]:*filename*  
/gd

**Default:**  
None

Use the /GENDEF option to create a module definition (.def) file.

#### Example

```
ilib /gd:sample.def sample.dll
```





## Using ILIB

The command above will create the module definition file `sample.def` from the DLL `sample.dll`.

### /GENIMPLIB

**Syntax:**

`/GENI[IMPLIB]:filename`  
`/gi`

**Default:**

None

Use the `/GENIMPLIB` option to create an import library/export object pair.

**Example**

```
ilib /gi sample.def
```

The command above will create an import library named `sample.lib` and an export object named `sample.exp` from the module definition file `sample.def`. However, if no exported symbols are contained, then `sample.lib` will not be produced.

### /HELP

**Syntax:**

`/H[ELP]`

**Default:**

None

Use `/HELP` to display a list of valid ILIB options. This option is equivalent to `/?`.

### /LIST

**Syntax:**

`/L[IST]:filename`

**Default:**

None

Use the `/LIST` option to generate a list file. If `filename` is not specified, ILIB will add the extension `.lst` to the input filename.

**Example**

```
ilib mylib /list:mylib.lst
```

The above command directs ILIB to place a listing of the contents of `mylib.lib` into the file `mylib.lst`. No path specification is given for `mylib.lst`. By default, the file created is put into the current directory.

**Note:** The `/LISTLEVEL` option is not supported in the Windows release of ILIB.

### /NOEXTDICTIONARY, /EXTDICTIONARY

**Syntax:**

`/NOE[XTDICTIONARY]`  
`/EXTD[ICTI]ONARY`

**Default:**

`/EXTDICTIONARY`

Use `/NOEXTDICTIONARY` to disable generation of the extended dictionary.

## Using ILIB

The extended dictionary is an optional part of the library that increases linking speed. However, using an extended dictionary requires more memory. The space reserved for the extended dictionary is limited to 64K. If ILIB reports an *out-of-memory* error, you may want to use this option. As an alternative, you can split large libraries into smaller libraries to use in linking.

### /OUT

**Syntax:**

/O[UT]:*filename*

**Default:**

None

Use the /OUT option to create or maintain an existing library. If *filename* is not specified, then the filename of the first input file specified on the command line will be used. ILIB will add the .lib extension.

**Examples**

```
ilib /out:newlib.lib mylib.lib sample.obj
```

The above command will create the library newlib.lib out of the objects in mylib.lib and sample.obj.

**Note:** Unless newlib.lib is specified as an input file, its contents will not be included in the library.

```
ilib /out newlib.lib mylib.lib
```

The above command will create the library newlib.lib by combining it with the library mylib.lib.

### /QUIET, /NOLOGO, /LOGO, /NOQUIET

**Syntax:**

/Q[UIET], /NOL[OGO]  
/LO[GO], /NOQ[UIET]

**Default:**

/LOGO, /NOQUIET

Use the /QUIET or /NOLOGO options to suppress the ILIB copyright notice.

### /WARN, /NOWARN

**Syntax:**

/W[ARN:*msgnum,msgnum*[...]]  
/NOW[ARN:*msgnum,msgnum*[...]]

**Default:**

None

Use the /WARN option to enable printing of the message number specified in the *msgnum* parameter.



## Chapter 46. Module Definition Files


A module definition file contains one or more module statements. These statements:


- Define various attributes of your executable output file
- Identify data and functions that are exported from your file

Use module definition files when you are creating a DLL, and did not define exports in your source files (using **#pragma export**, or the **\_Export** keyword). You can use the **EXPORTS** module statement to define exports, instead of defining exports in the source files. (Regardless of how the exports are defined, the linker requires a **.EXP** file to process them.)

**Note:** Module definition files cannot be used directly as input to the linker. You must use the library utility, **ILIB**, to generate the **.lib** and **.exp** files. Import libraries and export definition files can be used as input to the linker. See Part 8, “Managing Libraries” on page 517 for more information on **ILIB**.

When creating a module definition file, follow these rules:

- Use a **NAME** or **LIBRARY** statement to define the type of executable output you want. You can only use one of these statements, and it must precede all other statements in the module definition file.
- Begin comments with a semicolon (;). Any line that begins with a semicolon is ignored, as is any portion of a line that follows a semicolon.
- Enter all module definition keywords (for example, **NAME** and **LIBRARY**) in uppercase letters.
- Do not use module definition keywords or reserved words as text parameters to a statement (for example, you cannot use the **LIBRARY** statement to name a library **SHARED**, because **SHARED** is a keyword).  See “Reserved Words” on page 534 for a list of keywords and reserved words. See the section on calling conventions in the *Programming Guide* for details on name decoration.

 See “Module Statements” on page 536 for detailed descriptions of all statements.

## Reserved Words

### Example



```
;This is a module definition (.def) file for a  
;dynamic link library (DLL).
```

```
LIBRARY
```

```
;Identifies that the output produced by the linker will be a DLL
```

```
DESCRIPTION
```

```
'Sample DLL'
```

```
;Embeds a description in the DLL
```

```
STACKSIZE 1024
```

```
;Sets stack size to 1024
```

```
EXPORTS
```

```
;Makes data and functions defined inside the DLL available
```

```
;to other runtime modules
```

```
?Init @1
```

```
?Begin @2
```

```
?Finish @3
```

```
?Load @4
```

```
?Print @5
```

```
;The functions Init, Begin, Finish, Load, and Print can be called either
```

```
;by their entry name or by their ordinal positions (1, 2, 3, 4, or 5)
```

---

## Reserved Words

The following words cannot be used as text parameters to a module statement. For example, you cannot use these words as the names of functions defined with the EXPORTS statement, or to name a stub file with the STUB statement.

The words are either module definition keywords or are reserved by the linker.

## Reserved Words

**Note:** Although module definition keywords should always be entered in uppercase letters and only the uppercase forms are shown below, the mixed case and lowercase forms of these words are also reserved. For example, CONTIGUOUS, Contiguous, and contiguous are all reserved.

ALIAS	LIBRARY	PRELOAD
BASE	LOADONCALL	PRIVATE
CLASS	LONGNAMES	PROTECT
CODE	MAXVAL	PROTMODE
CONFORMING	MIXED1632	PURE
CONSTANT	MOVABLE	READ
CONTIGUOUS	MOVEABLE	READONLY
DATA	MULTIPLES	READWRITE
DECORATED	NAME	REALMODE
DESCRIPTION	NEWFILES	RESIDENT
DEVICE	NODATA	RESIDENTNAME
DEV386	NOEXPANDDOWN	ROBASE
DISCARDABLE	NOIOPL	SECTIONS
DOS4	NONAME	SEGMENTS
DYNAMIC	NONCONFORMING	SHARED
EXECUTE	NONDISCARDABLE	SINGLE
EXECUTEONLY	NONE	STACKSIZE
EXECUTE-ONLY	NONPERMANENT	STUB
EXECUTEREAD	NONSHARED	SWAPPABLE
EXETYPE	NOTWINDOWCOMPAT	SYSBASE
EXPANDDOWN	OBJECTS	TERMGLOBAL
EXPORTS	OLD	TERMINSTANCE
FIXED	ORDER	UNKNOWN
HEAPSIZE	OS2	VERSION
HUGE	PERMANENT	VIRTUAL
IOPL	PHYSICAL DEVICE	VIRTUAL DEVICE
IMPORTS		WINDOWAPI
IMPURE		WINDOWCOMPAT
INCLUDE		WINDOWS
INITGLOBAL		WRITE
INITINSTANCE		
INVALID		

## Module Statements Summary •BASE

### Summary of Module Statements

Figure 152. Module Statements Summary

Statement	Description	Parameters	Page
BASE= <i>address</i>	Set preferred loading address.	Loading address	536
DESCRIPTION ' <i>text</i> '	Describe the executable.	Descriptive text	537
EXPORTS [ <i>enm</i> =] <i>inm</i> [@ <i>ord</i> [ <i>keywrd</i> ]] [ <i>parms</i> ]	Define exported functions and data.	Entry name Internal name Ordinal position DECORATED CONSTANT Parameter size	537
HEAPSIZE <i>reserve</i> [, <i>commit</i> ]	Specify local heap size.	Virtual stack size Initial physical memory	539
LIBRARY [BASE= <i>address</i> ]	Identify output as dynamic link library (DLL). See detailed description for defaults of parameters.	Library name Loading address	540
NAME [BASE= <i>address</i> ]	Identify output as executable (EXE).	Application name Loading address	540
STACKSIZE <i>reserve</i> [, <i>commit</i> ]	Specify local stack size.	Virtual stack size Initial physical memory	541
STUB ' <i>filename</i> '	Add DOS executable file to module.	Filename to add	542
VERSION ' <i>file number</i> '	Adds string to executable.	Version number to add	542

### Module Statements

#### BASE

**Syntax:**  
BASE=*address*

**Parameters:**  
Loading address

Use the BASE statement to specify the preferred load address for the first load segment of the module.

This statement has the same effect as the /BASE linker option. If you specify both the statement and the option, the statement value overrides the option value.

## DESCRIPTION • EXPORTS

### DESCRIPTION

**Syntax:**

DESCRIPTION 'text'

**Parameters:**

Descriptive text

Use the DESCRIPTION statement to insert the specified text into the .EXE or .DLL file you are creating. The DESCRIPTION statement is useful for embedding source control or copyright information into your program or DLL.

The inserted text must be a one-line string enclosed in single quotation marks.

**Example**

Given the following line in a .def file,



```
DESCRIPTION 'Template Program'
```

the linker inserts the text Template Program into the .EXE or .DLL file.

### EXPORTS

**Syntax:**

```
EXPORTS  
[enm=] inm [@ord[keywrd]] [parms]
```

**Parameters:**

Entry name  
Internal name for function  
Ordinal position of function  
DECORATED|CONSTANT  
Size of function's parameters

Use the EXPORT statement when you are creating a dynamic link library (DLL) to define the names and attributes of data and functions exported from the DLL, and of functions that run with I/O hardware privilege.



You can also specify exports in your source code, using the **\_Export** keyword, or the **#pragma export** directive.

**Note:** Exported data and functions are those available to other .EXE or .DLL files. Data and functions that are **not** exported can only be accessed within your DLL, and cannot be accessed by other .EXE or .DLL files.

Give export definitions for functions and data in your DLL that you want to make available to other .EXE or .DLL files.

## EXPORTS

The EXPORTS keyword marks the beginning of the export definitions. Enter each definition on a separate line. You can provide the following information for each export:

- enm* The entry name of the data construct or function, which is the name other files use to access it. Always provide an entry name for each export. It is strongly recommended that you decorate each entry name, to ensure that the correct function is linked in.
- inm* The internal name of the data construct or function, which is its actual name as it appears *within* the DLL. If specified, the internal name must be decorated. If you do not specify an internal name, the linker assumes it is the same as *enm*.
- ord* The data construct or function's ordinal position in the module definition table. If you provide the ordinal position, the data construct or function can be referenced either by its entry name or by the ordinal. It is faster to access by ordinal positions, and may save space.
- keywrd* You can specify one of the following two values. **DECORATED** is the default.
- |                  |                                                                               |
|------------------|-------------------------------------------------------------------------------|
| <b>DECORATED</b> | The name should be left as-is. No decoration is applied to the function name. |
| <b>CONSTANT</b>  | Indicates that the export entity is a variable, not a function.               |
- You cannot specify both values.
- parms* The total size of the function's parameters, as measured in words (bytes divided by two). This field is required only if the function executes with I/O privilege. When a function with I/O privilege is called, Windows consults *parms* to determine how many words to copy from the caller's stack to the stack of the I/O-privileged function.

### Example

The following example defines three exported functions:

- SampleRead
- StringIn
- CharTest



```
EXPORTS
?SampleRead = ?read2bin @8
?StringIn = ?strl @4
?CharTest 6
```



## HEAPSIZE

The first two functions can be accessed either by their exported names or by an ordinal number. Note that in the module's own source code, these functions are actually defined as `read2bin` and `str1`, respectively. The last function runs with I/O privilege, and so has *parms* (the total size of the parameters) defined for it as six words.

## HEAPSIZE

### Syntax:


`HEAPSIZE reserve[,commit]`

### Parameters:


Virtual stack size

Initial physical memory

Use the `HEAPSIZE` statement to define the size of the application's local heap in bytes.

You can enter any positive integer for the heap size.  See “Specifying Numeric Arguments” on page 199.

*reserve* indicates the total virtual address space reserved. *commit* sets the amount of physical memory to allocate initially. When *commit* is less than *reserve*, memory demands are reduced, although execution time may be slower.

Values specified by the `/HEAP` linker option take precedence over the `HEAPSIZE` statement.  See “/HEAP” on page 209 for more information.

### Example

Given the following line in a `.def` file,



```
HEAPSIZE 4000
```

the linker sets the local heap to 4000 bytes.

## LIBRARY •NAME

### LIBRARY

**Syntax:**

LIBRARY [BASE=*address*]

**Parameters:**

Loading address

Use the LIBRARY statement to identify the output file as a dynamic link library (DLL), and optionally define the name, library module initialization, and library module termination.

You can also identify the output file as a DLL with the /DLL option.

If you use the LIBRARY statement in your module definition (.def) file, it must be the first statement in the .def file, and you cannot use the NAME statement.

If you specify both the BASE parameter in the LIBRARY statement and the BASE statement, the BASE statement takes precedence.

**Example**

The following example assigns the name calendar to the dynamic link library (DLL).

```
LIBRARY calendar
```

### NAME

**Syntax:**

NAME [BASE=*address*]

**Parameters:**

Application name

Loading address

Use the NAME statement to identify the output file as an executable program (.EXE file), and optionally define the name and type of the .EXE file.

You can also identify the output file as an .EXE file with the /EXEC option.

If you use the NAME statement in your module definition (.def) file, it must be the first statement in the .def file, and you cannot use the LIBRARY statement.

If you specify a loading address with the BASE parameter and the /BASE option is also specified, the address specified by the option takes precedence.

## STACKSIZE

### Example

The following example assigns the name `calendar` to the executable program.



```
NAME calendar
```

## STACKSIZE

### Syntax:

```
STACKSIZE reserve[,commit]
```


### Parameters:

Virtual stack size


Initial physical memory

Use `STACKSIZE` to set the stack size (in bytes) of your program. The size must be an even number, from 0 to `0xFffffe`. If you specify an odd number, it is rounded up to the next even number.

*reserve* indicates the total virtual address space reserved. *commit* sets the amount of physical memory to allocate initially. When *commit* is less than *reserve*, memory demands are reduced, although execution time may be slower.

Values specified by the `/STACK` linker option take precedence over the `STACKSIZE` statement.  See “`/STACK`” on page 214 for more information.

If your program generates a stack-overflow message, use the `STACKSIZE` statement to increase the size of the stack.

If your program uses the stack very little, you can save some space by decreasing the stack size.  See “Controlling Stack Allocation and Stack Probes” on page 102 for more information.

The `STACKSIZE` statement is equivalent to the `/STACK` linker option. If you specify both the statement and the option, the statement value overrides the option value.

### Example



The following example allocates 4K of local-stack space:

```
STACKSIZE 4096
```

## STUB • VERSION

### STUB

**Syntax:**

STUB '*filename*'

**Parameters:**

Name of file to add

Use the STUB statement to add a DOS .EXE file to the beginning of your .EXE or .DLL. The stub function is then invoked whenever your .EXE or .DLL file is run under DOS. Typically, the stub displays the message that the program cannot run in DOS mode, and ends the program.

If you do not use the STUB statement, the linker adds its own standard stub for this purpose.

The linker searches for the filename you specify as the stub as follows:

1. In the directory you specify, or in the current directory if you did not give a path
2. In the directories listed in the PATH environment variable

**Example**

The following example adds the DOS .EXE file STOPIT.EXE to the beginning of the file you are creating. STOPIT.EXE runs whenever your file is run under DOS.

```
STUB 'STOPIT.EXE'
```

### VERSION

**Syntax:**

VERSION '*file number*'

**Parameters:**

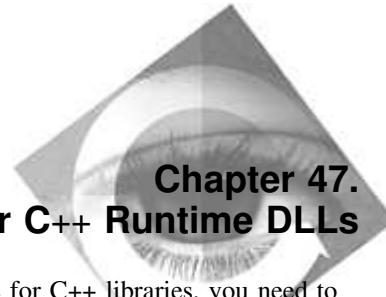
Version number to add

Use the VERSION statement to add a version number to the header of the run file.

**Example**

The following example adds the text 'VERSION 2.3' to the executable.

```
VERSION '2.3'
```



## Chapter 47. Packaging the VisualAge for C++ Runtime DLLs

If your application uses functions from the VisualAge for C++ libraries, you need to ensure the code for those libraries is always available to your application. You cannot ship the VisualAge for C++ DLLs themselves with your application because of the product licensing agreement and because if more than one application included the VisualAge for C++ DLLs, but at different levels, at least one application would be using the wrong level.


If you are shipping your application to other users who do not have access to the library DLLs, you can use one of three methods to include the VisualAge for C++ library code:


1. Statically bind every module to the library (.LIB) files. Compile with /Gd-, which is the default.

This method increases the size of your modules and slows the performance because the library environment has to be initialized for each module. Having multiple library environments also makes signal handling, file I/O, and other operations more complicated.

2. Create your own runtime DLLs.

This method provides one common runtime environment for your entire application. It also lets you apply changes to the runtime library without relinking your application, meaning that if the VisualAge for C++ DLLs change, you need to rebuild only your DLL.

For a description of how to build your own runtime DLL, or subsystem runtime DLL,  see the *Programming Guide*.

3. Use the DLL rename utility, DLLRNAME, to rename the VisualAge for C++ library DLLs. This utility also changes the names in your executable files that call the DLLs. DLLRNAME is part of the VisualAge for C++ product, and is  described in “Using the DLLRNAME Utility” on page 544.

## Packaging the Runtime DLLs

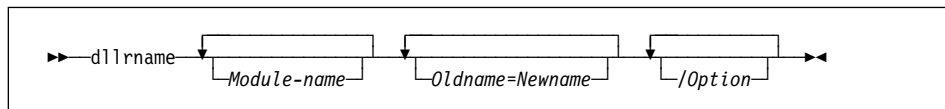
---

### Using the DLLRNAME Utility

To use the DLL rename utility, build your application using the import libraries provided with VisualAge for C++ (compiling with the /Gd+ option). Then, before you ship your application:

1. Copy the VisualAge for C++ DLLs that your application uses into a working directory.
2. Run the DLL rename utility, DLLRNAME, against your executable files and your working copies of the VisualAge for C++ DLLs. The utility will rename the DLLs as well as all internal names that need to be changed as a result.

The syntax for the dllrname command is:



#### Module Names

The list of module names includes the VisualAge for C++ library DLLs your application uses, along with the EXEs and DLLs that reference them. They must be present in the current directory, unless their paths are specified. Case is disregarded when matching names.

**Note:** It is important that you include all the modules in your application in this list, since the names by which the modules reference the VisualAge for C++ library DLLs must also be changed in the modules themselves.

#### Name Changes

You specify the list of the name changes to be made by indicating `Oldname=Newname` on the DLLRNAME command line.

*Oldname* is the name of the VisualAge for C++ library DLL as it was shipped with VisualAge for C++.

*Newname* is the name under which you will be shipping the VisualAge for C++ library DLL with your application.

**Note:** The DLLRNAME utility requires that *Oldname* and *Newname* have the same number of characters.

For example, to rename the VisualAge for C++ library DLL `CPPWS35.DLL` to `MYLIBRY.DLL` in the modules `myprog.exe` and `mydll.dll`, specify the following on the DLLRNAME command line:

```
DLLRNAME myprog.exe mydll.dll CPPWS35=MYLIBRY
```

---

### How DLLRNAME Works

All modules (.EXE and .DLL files) that use other DLLs contain records specifying a set of external file names that are needed to run the module. The DLLRNAME utility manipulates only these records; it does not modify your executable code.

One of the external names specified in a module is the name of the module itself. The name of module, as it appears in its own internal records, is called its *internal name*. You specify this name with the NAME or LIBRARY record in a DLL module definition (.DEF) file when you link the module. In the case of an .EXE file, the loader essentially ignores the internal name. For a .DLL, its internal name must match its filename; otherwise the loader will refuse to load the DLL. By default, DLLRNAME will also change the filename of a DLL if it changed its internal name.

The rest of the external names specified in a module are the names of the DLLs to be loaded when the Windows loader loads the module. All of these DLLs must be loaded for the Windows loader to successfully load the module. If you specify any of these required DLLs for rename on the DLLRNAME command line, DLLRNAME also changes the names of the required DLLs in the module itself.

The DLLRNAME utility accepts 32-bit Microsoft Portable Executable (PE) executables. It will not change DOS or OS/2 executables, or any other file with a different format.

### What DLLRNAME Will Not Do

The DLLRNAME utility will not:

- Modify DLLs named explicitly in your executable code. This is important to remember if you use the Windows LoadLibrary() API or the VisualAge for C++ \_loadmod function to load a required DLL. You must modify your code to load the DLL using its modified name.
- Modify DOS or OS/2 executables.
- Change the name of a DLL to a name of different length.

### Other Uses for DLLRNAME

The DLLRNAME utility can also be used to:

- Rename your own DLLs so that you can have multiple versions of your application resident on the same machine for testing purposes.
- Obtain a report that lists all the DLLs used by a module. Simply invoke the DLLRNAME utility without specifying any DLL name changes.

## Packaging the Runtime DLLs

---

### DLLRNAME Options

The following options control the operation of the DLLRNAME utility:

**/H or /?**    Display help  
**/N**            Do not rename DLL  
**/Q**            Suppress display of logo  
**/R**            Do not generate report

**Note:** You can specify options using the slash form (/R) or the dash form (-R).

#### /H (Help)

Syntax:	Default:
/H	None
/?	

Specify the /H or /? option on the DLLRNAME command line to see a short online help on the dllrname command syntax and options.

If you do not specify this option, the default is not to display any online help.

#### /N (Do Not Rename DLL)

Syntax:	Default:
/N	Rename all DLLs

Specify the /N option on the DLLRNAME command line to instruct the DLLRNAME utility not to rename any DLLs that appear in both the modules list and the list of name changes.

If you do not specify this option, the default is to rename any DLLs that appear in both the modules list and the list of name changes.

#### /Q (Do Not Display Logo)

Syntax:	Default:
/Q	Display logo and copyright notice

Specify the /Q option on the DLLRNAME command line to suppress the display of the logo and copyright notice for the DLLRNAME utility.



## Packaging the Runtime DLLs

If you do not specify this option, the default is to display the logo and copyright notice.

### /R (Do Not Generate Report)

**Syntax:**

/R

**Default:**

Generate report

Specify the /R option on the DLLRNAME command line to suppress the generation of a report detailing the name changes.

If you do not specify this option, the default is to generate a report detailing the name changes.

---

## An Example



If you compiled your application using the following command lines:

```
ICC /Gd+ /Ge- /FeA.DLL A.C B.C C.C D.C A.DEF
```

```
ICC /Gd+ /FeE.EXE E.C F.C G.C H.C A.LIB
```

your application would be made up of the files A.DLL and E.EXE. Since you specified the /Gd+ compile option, your application also requires the file CPPWS35I.DLL from VisualAge for C++.

To obtain a renamed copy of CPPWS35I.DLL that you may ship with your application, use the following set of commands:

```
REM Get a working copy of the VisualAge for C++ library DLL
COPY D:\IBMCPW\DLL\CPPWS35I.DLL
```

```
REM Change all the names
DLLRNAME A.DLL E.EXE CPPWS35I=MYDLL35I
```

```
DLLRNAME CPPWS30.DLL CPPWS35I=MYDLL35I
```

These commands will change A.DLL and E.EXE so that they will now require MYDLL35I.DLL instead of CPPWS35I.DLL. DLLRNAME will also rename CPPWS35I.DLL to MYDLL35I.DLL.

## Packaging the Runtime DLLs

The following is the text of the report generated by the DLLRNAME utility:

```
> dllrname a.dll e.exe cppws35i.dll cppws35i=mydll35i
```

```
Licensed Materials - Property of IBM  
IBM C/C++ Tools Version 2.0 - DLL Rename Utility  
(C) Copyright IBM Corp., 1993, All Rights Reserved  
US Government Users Restricted Rights - Use, duplication or disclosure  
restricted by GSA ADP Schedule Contract with IBM Corp.
```

Processing file a.dll.

```
1 external names in file a.dll have been left unchanged.  
2 names found in file a.dll.  
Executable name a has been left unchanged.  
Imported DLL name CPPWS35I has been changed to mydll35i.  
Imported DLL name KERNEL32 has been left unchanged.
```

Processing file e.exe.

```
1 external names in file e.exe have been left unchanged.  
2 names found in file e.exe.  
Executable name e has been left unchanged.  
Imported DLL name CPPWS35I has been changed to mydll35i.  
Imported DLL name KERNEL32 has been left unchanged.
```

Processing file cppws35i.dll.

```
1 external names in file cppws35i.dll have been left unchanged.  
2 names found in file cppws35i.dll.  
Executable name CPPWS35I has been changed to mydll35i.  
Imported DLL name KERNEL32 has been left unchanged.  
File cppws35i.dll has been renamed to mydll35i.DLL to match internal DLL name.
```

Complete. 0 error(s) detected.

---

## Part 9. Using the Data Access Builder

This part of the *User's Guide* describes the Data Access Builder, an application development tool you can use to bring your existing relational data into the world of object-oriented applications. This part also describes the installation, configuration, and use of the ODBC drivers included with IBM VisualAge for C++ for Windows.

---

<b>Chapter 48. Introduction</b>	551
Overview	551
Setting Up a Data Access Builder WorkFrame Project	552
Starting Data Access Builder	553
The Data Access Builder Window	554
Customizing the Display of Mappings	556
Registering Database Products	556
Steps Involved in Creating Classes	557
 <b>Chapter 49. Creating Table and Class Objects</b>	 559
Accessing Database Tables/Views	559
Selecting the Code Generation Options	561
Identifying the Data Access Builder Icons	565
Generating Different Kinds of Classes for the Same Table/View	566
Deleting Class and Table/View Objects	567
Viewing the Table/View Settings	568
 <b>Chapter 50. Generating the Source Code</b>	 573
Customizing the Class Mapping	573
Generating the Class Source Code	578
Saving a Data Access Builder Session	579
Viewing Generated Files	580
Using the Generated Files	580
 <b>Chapter 51. Getting Started with ODBC</b>	 581
About ODBC Drivers	581
Installing the ODBC Drivers	582
Windows 3.x Run-time Support	582
Error Messages	582
Locking and Isolation Levels	584
 <b>Chapter 52. Supported ODBC Drivers</b>	 587
DB2 Driver	587

## Using the Data Access Builder

Oracle 7 Driver . . . . .	588
Sybase System 10 Driver . . . . .	593

---



## Chapter 48. Introduction

The topics covered in this chapter include:

- A general overview of the features of the Data Access Builder
- Instruction on how to launch the tool
- An introduction to the elements of the Data Access Builder window
- A brief description of the steps involved in creating your own data access classes

---

### Overview

Data Access Builder is an application development tool you use to create database access classes customized for your existing relational database tables. It allows you to create object-oriented applications quickly and reliably by generating the source code for you.

These database access classes can be used directly in your C++ programs, or you can generate visual parts and import them directly into Visual Builder. By using Visual Builder to connect them to the GUI, or to other parts, you can create high-quality applications quickly and efficiently. Additionally, you can generate language-independent IDL source that allows you to access your relational data as SOM objects.

Some of the key features of Data Access Builder are:

- **Visual mapping of tables to classes**  
The Data Access Builder graphically displays the mappings of your database tables to the object classes. This view allows visual editing and uses icons for tables and classes, and arrows to show the mappings.
- **Quick or custom mapping**  
The Data Access Builder offers a quick map feature that allows you to do a column-to-attribute direct mapping. Users can then customize their classes to suit their needs.
- **Connection and transaction services**  
Separate services are provided for connection and disconnection from your databases. In addition, commit and rollback operations are provided to handle transaction services.
- **Data manipulation operations**  
Generated classes customized to your data help you perform common database tasks such as adding, retrieving, updating, and deleting data. Additional services

## Using the Data Access Builder

are provided for the selection and retrieval of a group of objects from the datastore, which can then be manipulated using the collection class library.

- **Multiple connections and multiple access methods**

The generated class library provides support for multiple connections to the same and/or multiple datastores. Multiple database-access methods allow you to access various database products: direct support for DB2 using either embedded SQL or the DB2 Call Level Interface (DB2 CLI); Open Database Connectivity (ODBC) support for access to many database products using the ODBC CLI and appropriate database driver.

## Supported Database Products

The Data Access Builder supports DB2 version 2.1 directly through your choice of embedded SQL or the DB2 Call Level Interface (DB2 CLI). Access to a multitude of database products is also available through the ODBC driver manager and related drivers. With this release of VisualAge for C++, *official support* is only extended to ODBC CLI access to Sybase System 10, Oracle 7, and DB2 through their respective ODBC drivers. For more information on ODBC support, refer to Chapter 51, “Getting Started with ODBC” on page 581 and Chapter 52, “Supported ODBC Drivers” on page 587.

— **Attention!** —

Although any database product supported by a compatible ODBC 2.0 driver can be accessed reliably through Data Access Builder's ODBC CLI support, *only the ODBC drivers for the aforementioned officially-supported database products can be used in the development of applications for which IBM Service is requested.*

---

## Setting Up a Data Access Builder WorkFrame Project

To simplify the creation of applications, **Project Smarts** are used to create WorkFrame project templates. There are two **Project Smarts** templates that are specially customized for applications that have data access components. One project template is provided for standard C++ and SOM applications which use data access classes; the second project template is for data access applications that will be created using the Visual Builder.

To start the **Project Smarts** tool:

- From the Windows interface, double-click on the **WorkFrame** icon in the **VisualAge for C++** program group/folder.
- From a command prompt, type `ismarts` and press **Enter**.

## Using the Data Access Builder

Then select **Data Access** or **Visual Data Access** from the **Project types** list box and click on the **Next>>** push button. **Project Smarts** then guides you through the process of creating a WorkFrame project customized to your specifications.

---

### Starting Data Access Builder

To start Data Access Builder you can use any of the following methods:

- From the Windows interface:
    - WindowsNT:
      - Double-click on the **Data Access Builder** icon in the **VisualAge for C++** program group.
    - Windows95:
      1. Click on the **Start** button.
      2. Select **Programs** from the pop-up menu.
      3. Select **VisualAge for C++** from the **Programs** cascaded menu.
      4. Select **Data Access Builder** from the resulting cascaded menu.
  - From the WorkFrame tool bar:
    - Click on the **Data Access Builder** icon.
  - From a WorkFrame project or any WorkFrame-integrated tool:
    - Select **Database** from the **Project** menu, or
    - Select **Database** from the window control menu.
  - From a command prompt:
    - Change to your working directory; type **idata** and press **Enter**.
- Note:** When you start Data Access Builder from a command prompt, the current directory also becomes the target directory for the files generated by the tool. See “Specifying Default Generation Options” on page 563 for information on changing the default target directory once the tool is started.

### Opening a Previously Saved Data Access Builder Session

Data Access Builder allows you to save your session at any point in your use of the tool (see “Saving a Data Access Builder Session” on page 579). To open a previously saved session:

- From the Data Access Builder window:
  1. Choose **Open** from the **File** menu.
  2. Type the name of the session you want to restart in the **Filename** entry field.
  3. Choose **OK** to open the session.
- From a WorkFrame project:

## Using the Data Access Builder

- Double-click on any \*.dax file icon, or
  - Click mouse button two on any \*.dax file icon and select **Database** from the pop-up menu.
- From a command prompt:
  - Type `idata filename.dax` and press **Enter** (substitute `filename.dax` with the filename you entered when you previously saved your session)

## Migrating a Previously Saved Data Access Builder Session

You can migrate .dax files from a previous release of VisualAge for C++ to use with this release. You import these files the same way you open a previously saved session (see “Opening a Previously Saved Data Access Builder Session” on page 553). *However*, there are some things of which you must be aware. For full details on migrating files, please refer to one of the following:

- The Data Access Builder *How do I...* help information:  
Select **How do I...** from the **Help** menu and then select **Migrating My Existing .DAX Files** from the list of topics.
- The "Data Access Builder" section of the "Making Your Programs Portable" chapter in the *Programming Guide*.

---

## The Data Access Builder Window

Figure 153 on page 555 shows the window that is displayed when you start the Data Access Builder.



## Using the Data Access Builder

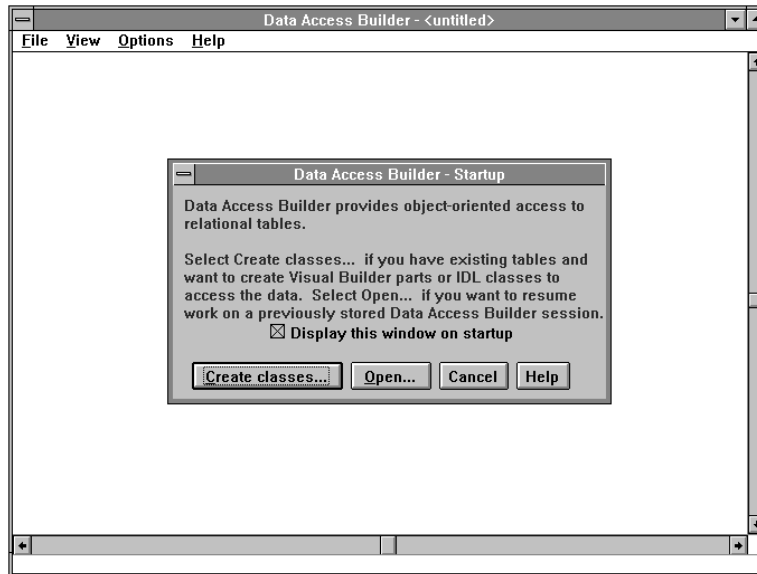


Figure 153. Data Access Builder window with Startup dialog box displayed

Note the following items that make up the Data Access Builder user interface:

- The **Startup** window is displayed (default setting).
- The area behind the **Startup** window, in the main window, is referred to as the *client area*. This is where the table-to-class mappings that you create will be displayed.
- Mappings can often take up more space than can be displayed on the screen; the scroll bars and arrows can be used to reveal parts of your mappings that do not fit in the window.

### Startup Window

The **Startup** window appears when you first start the Data Access Builder tool. This window allows you to create, or open a previously created, table-to-class mapping without having to make the selection from the **File** menu.

You can choose whether or not this window appears when you start the tool. The **Startup** window can be toggled on or off by selecting **Startup Window** from the **Options** menu. A checkmark appears in front of the menu item when the item is selected, and disappears when deselected. Any changes you make to this item take effect the next time you start the Data Access Builder tool.

## Using the Data Access Builder

---

### Customizing the Display of Mappings

The **View** menu allows you to change how the mappings are displayed on the screen. Menu items in the **View** menu allow you to choose whether mappings are displayed horizontally or vertically across the screen, and the size of the icons used in the mappings.

These settings have been largely provided to allow you to view as much of your mappings as possible; they have also been provided to help you choose the most appropriate settings for your particular screen resolution.

#### Icon Size

To set the table/view and class icon size, select **Icon size** from the **View** menu and from the resulting cascaded menu, select the size you want.

#### Horizontal/Vertical Orientation

Because windows are generally wider than they are tall, selecting a horizontal view will usually permit more mappings to be visible on the screen at once. However, you may prefer to resize the Data Access Builder window to a tall thinner size to reveal more of the general screen, in which case a vertical view would be preferable. To set the display orientation, select either **Vertical** or **Horizontal** from the **View** menu. A checkmark appears next to the option you choose.

---

### Registering Database Products

The first time Data Access Builder is started, it searches your workstation for all recognized database products and stores this information in its initialization file. However, if after your initial use of Data Access Builder you upgrade, add, or remove database products, you must register the changes so that Data Access Builder will allow you to properly access the installed database products on your system when creating classes. To update the list of supported database products available on your system, select **Register Databases** from the **Options** menu.

---

## Steps Involved in Creating Classes

The general steps involved in creating classes for your existing database tables or views are as follows:

1. Accessing the database tables/views by:
  - a. Connecting to the database
  - b. Selecting the desired table(s) and/or view(s)
2. Selecting the desired access method and generated class type
3. Optionally customizing the table-to-class mapping
4. Generating the code
5. Saving and exiting your session

## Using the Data Access Builder

## Chapter 49. Creating Table and Class Objects

This chapter explains how to create table/view and class objects. Topics covered in this chapter include:

- Connecting to the database and selecting the desired table(s) and/or view(s)
- Selecting the desired code generation options and setting default options
- Identifying table/view and class icons
- Deleting table and class objects
- Viewing the table/view settings
- Changes in the database table/view definition

---

### Accessing Database Tables/Views

**Note:** For databases accessed through ODBC, please make sure your database system is started before attempting to access your database tables or views.

To access your database tables or views:

1. Click on the **Create classes** push button from the **Startup** window or select **Create classes** from the **File** menu. The **Create classes** dialog box is displayed on the screen (refer to Figure 154).

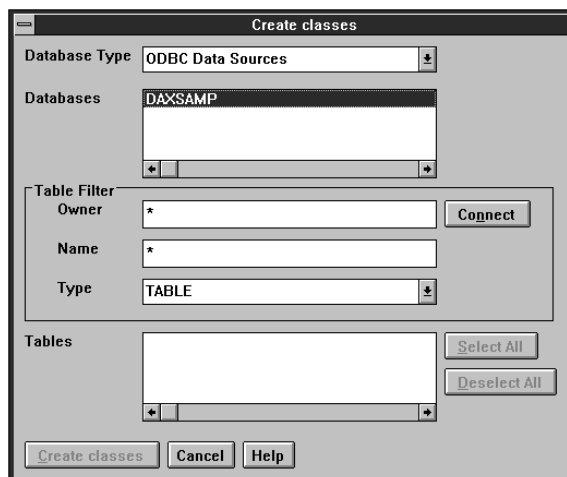


Figure 154. Create Classes dialog box.

## Using the Data Access Builder

2. If the desired database product is not listed in the **Database Type** field, click on the down arrow to the right of the field to display a list box of all the supported database products on your system, and select a product from the list.

**Note:** If you have installed a supported database product on your workstation and it does not appear in the **Database Type** list box, please refer to “Registering Database Products” on page 556.

3. The **Databases** list box displays all accessible databases for the selected **Database Type**. To connect to the database:
  - a. Select the desired database name from the list.
  - b. You may optionally specify which tables will be displayed in the **Tables** list box by using the **Table Filter**; tables can be filtered by table owner, name, or type.
    - Type an owner name in the **Owner** entry field and/or type a table name in the **Name** entry field. Owner name and table name are case sensitive and the use of the wildcard characters ? and \* (respectively, single character and multiple character wildcards) is permitted.
    - Select a table type by clicking on the down arrow to the right of the **Type** field and selecting a table type from the list displayed in the drop-down list box.
  - c. Finally, click on the **Connect** push button. (Alternatively, to connect to a database and accept the filter information as it appears, you can just double-click on the desired database name).
4. At this point, a **Logon** dialog box may appear on the screen. If this occurs, enter your database user id and password and click on **OK**.
5. A list of all available tables and views for the selected database will be displayed in the **Tables** list box (subject to any specified filter). Select the desired tables(s) and/or view(s) from the list; the **Select All** and **Deselect All** push buttons can be used to help you make your selection. (See Figure 155 on page 561.)

## Using the Data Access Builder

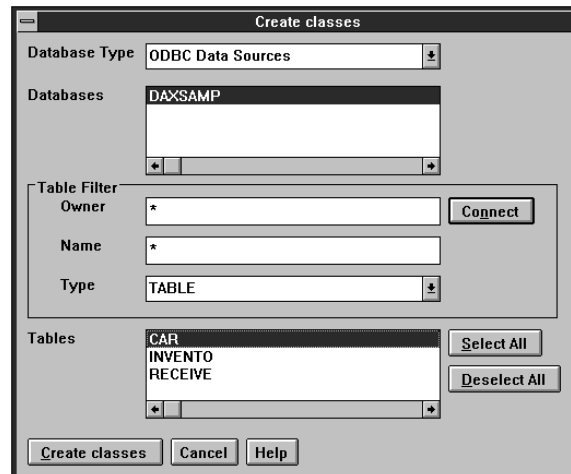


Figure 155. Create Classes dialog box with database and table selected

- Click on the **Create classes** push button to create a class for each table or view selected; or click on the **Cancel** push button to close the **Create classes** dialog box without creating any classes.

**Note:** Alternatively, double-clicking on the desired table or view name is equivalent to selecting that item and clicking on the **Create classes** push button.

---

### Selecting the Code Generation Options

When you have followed the steps in “Accessing Database Tables/Views” on page 559, the **Create Class - Generate Options** dialog box appears (see Figure 156 on page 562).

**Note:** If the **Create Class - Generate Options** dialog box does not appear, then the **Use these defaults without prompting** option has been selected, in which case, the previously specified default options are used (see “Specifying Default Generation Options” on page 563).

## Using the Data Access Builder

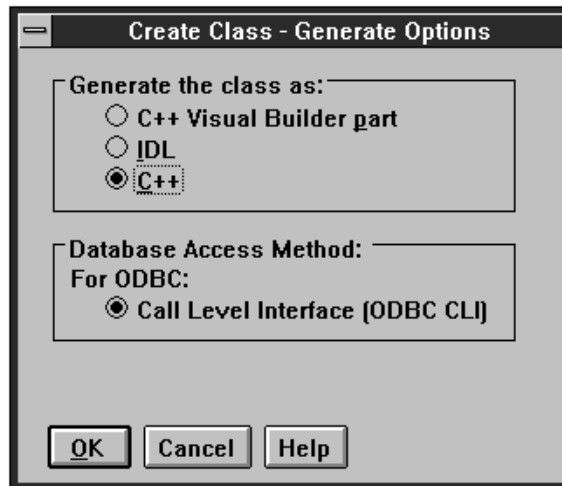


Figure 156. Create Classes - Generate Options dialog box

To create the class object for the table(s) and/or view(s) you selected:

1. Select the type of source code that will be generated by the Data Access Builder by clicking on the button for **C++ Visual Builder part**, **IDL**, or **C++**. If you want to generate more than one type of source code for a table mapping, refer to “Generating Different Kinds of Classes for the Same Table/View” on page 566.
2. Select the database access method that will be used by the Data Access Builder by clicking on the button for DB2 **Embedded SQL** or **Call Level Interface (CLI)**, or for **ODBC CLI**.

**Note:** Database access methods that are not valid for the selected database do not appear or appear shaded. For example, a Sybase table is only supported through the ODBC CLI access method, and therefore the DB2 **Embedded SQL** and **Call Level Interface (CLI)** access methods would not appear.

3. Click on **OK** to accept your selected options and create table/view-to-class mappings for each table or view you selected. Or click on **Cancel** to cancel the creation of classes and return to the Data Access Builder window.

Figure 157 on page 563 is an example of a view-to-class mapping that would be displayed in the client area after performing the above steps.



## Using the Data Access Builder

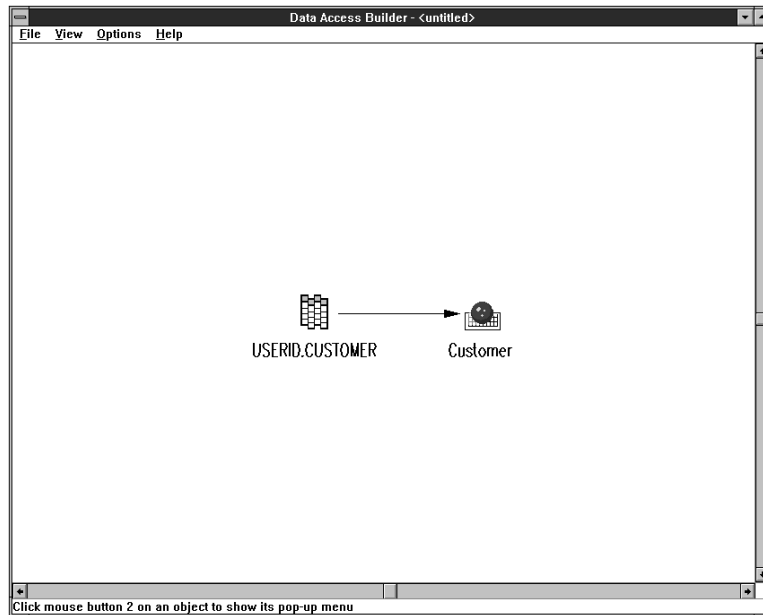


Figure 157. A view-to-class object mapping

### Specifying Default Generation Options

If you wish to customize which options are selected as defaults:

1. Select **Generate Options...** from the **Options** pull-down menu. The **Generate Options** dialog box will be displayed (see Figure 158 on page 564).
2. Select the **Generate the class as** and **Database Access Method** options you would like to set as defaults.

## Using the Data Access Builder

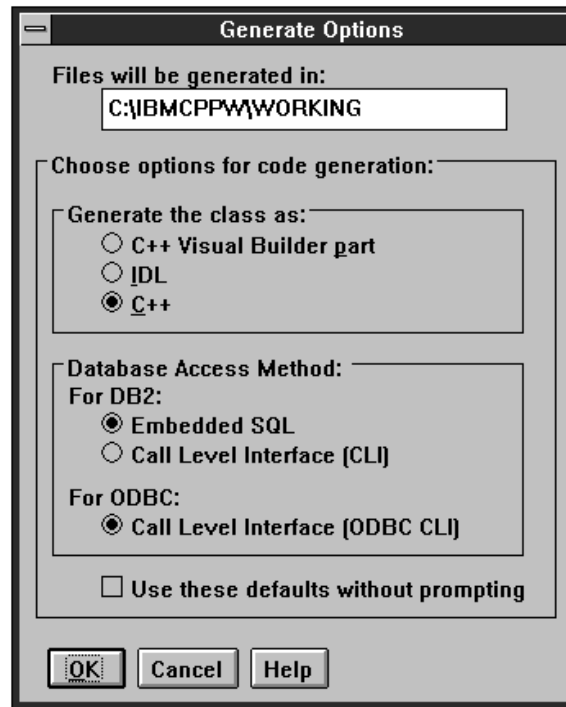


Figure 158. Generate Options dialog box

3. If you will be generating a series of classes that use the same options, and you do not want the **Create Class - Generate Options** dialog box to appear each time you create a new class, click in the **Use these defaults without prompting** check box to set a check mark.
4. Additionally, this dialog box allows you to change the default directory in which generated files are saved. In the entry field below the text **Files will be generated in**, enter the full path to the directory in which you would like the generated files saved.
5. Click on **OK** to accept your selections as the default settings.

---

## Identifying the Data Access Builder Icons

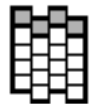
Mappings created by the Data Access Builder tool will be displayed in the Data Access Builder client area using the following icons:

### Database Table Icons

There are two kind of objects that represent database entities: tables and views. Tables consist of columns and rows. The table icon visually represents this as a grid. A view is a collection of columns from one or more tables. The view icon visually represents this by skewing the columns of a table to imply that the columns may not all be from the same table.



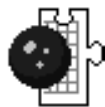
Database table icon



Database view icon

### Class Object Icons

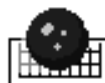
Class object icons consist of a blue sphere, representing the object, and a table piece, representing the generated class type. When the code for the class object has not been generated, the table piece is black-and-white. When the source code for the class has been generated (see “Generating the Class Source Code” on page 578), the table piece changes to yellow (for Visual Builder parts), green (for IDL), or purple (for C++ classes).



C++ Visual Builder part icon



IDL object icon



C++ class object icon

---

### Generating Different Kinds of Classes for the Same Table/View

Once a table or view is displayed in the client area, if you wish to use that table in another mapping (that is, create a different class object), you must select **Create classes** from the table or view object's pop-up menu, not from the **File** menu. Only use **Create classes** from the **File** menu to create a mapping for a table or view that is not displayed in the client area.

Some reasons why you may want to map more than one class object to a table are:

- To generate different types of source code for the same class object, for example, C++ and IDL.
- To create different class mappings for a table; for example, you may want to create class objects that contain different subsets of columns from the same table or view.

To create another class object for the same table (or view):

1. Click mouse button two on the table/view object and select **Create class** from the pop-up menu. The **Create Class - Generate Options** dialog box is displayed.

**Note:** If the **Create Class - Generate Options** dialog box does not appear then the **Use these defaults without prompting** option has been selected, in which case, the previously specified default options are used (see “Specifying Default Generation Options” on page 563).

2. Select the options you want (refer to “Selecting the Code Generation Options” on page 561), and click on the **OK** push button. An additional class object is mapped to the table. The class name always defaults to the table name, but if there is already a class object with that name, Data Access Builder creates a unique name by suffixing a number to the name.
3. To change the name of the new class or to customize the default mapping, refer to “Customizing the Class Mapping” on page 573.

In Figure 159 on page 567 we see an example of a table object mapped to more than one class object. In this case, class objects were created for the various types of code (C++, IDL, Visual Builder) that can be generated.

## Using the Data Access Builder

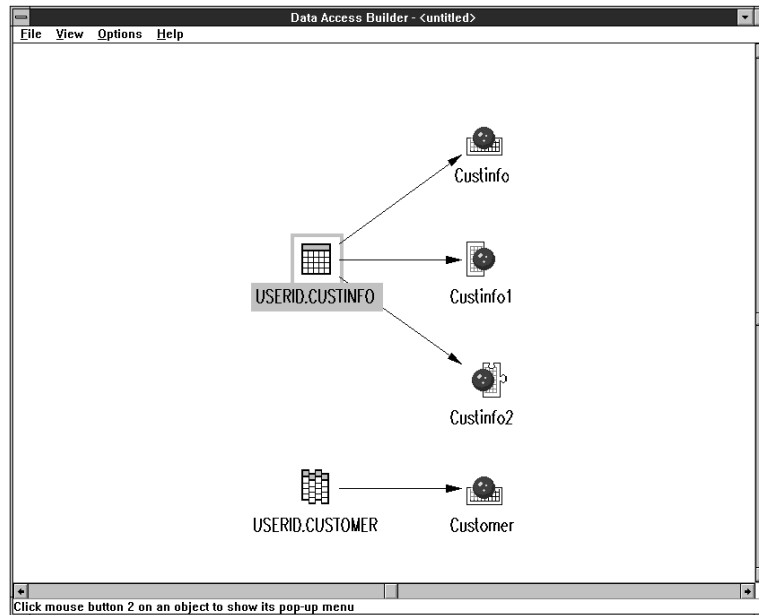


Figure 159. Multiple class objects created for the same table

---

### Deleting Class and Table/View Objects

If you wish to delete a class object, click mouse button two on the class object's icon and select **Delete** from the pop-up menu. The class object and all its settings will be deleted from the client area. Any source files already generated for that class will not be erased.

If you wish to delete a table or view object from the client area, click mouse button two on the table or view object icon and select **Delete** from the pop-up menu. The table or view *and any class objects mapped to it* are deleted from the client area. *This action deletes the entire mapping from the client area.*

#### Attention!

By default, you will be prompted before tables/views or class objects are deleted; however, you will *not* be asked to confirm any deletion if **Confirm on delete** is not checked in the **Options** pull-down menu.

## Using the Data Access Builder

---

### Viewing the Table/View Settings

To open the **Settings** notebook for a table or a view object in the client area, click mouse button two on the table/view object and select **Open settings** from the pop-up menu. Alternatively, just double-clicking on the table or view object will also bring up the **Settings** notebook.

The **Settings** notebook shows you details about the table or view as it is defined in the database. This information is read-only: it cannot be changed from the **Settings** notebook. You cannot use the Data Access Builder tool to create tables or views, or to change their definitions. This must be done using the database administration facility.

The **Settings** notebook for a table has three pages: Table, Columns, and Foreign keys. The **Settings** notebook for a view has two pages: View, and Columns. A view's **View** and **Columns** pages are functionally equivalent to a table's **Table** and **Columns** pages.

### Table/View Page

The table/view's **Settings** notebook opens on the **Table/View** page (see Figure 160).

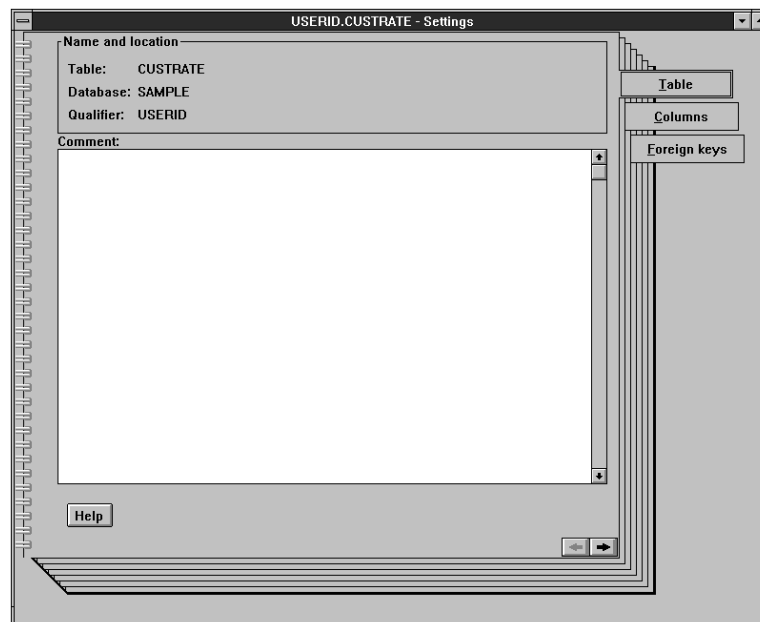


Figure 160. Table Settings notebook - Table page

## Using the Data Access Builder

The **Table** page of a table's **Settings** notebook (or the **View** page of a view's **Settings** notebook) contains the following information:

- The name of the table/view
- The name of the database that contains the table/view
- The user ID of the creator of the table/view (which is used as a qualifier for the table/view name when it is displayed in the client area)
- Any comments that the creator included with the table

### Columns Page

Clicking on the **Columns** tab displays the **Columns** page (see Figure 161).

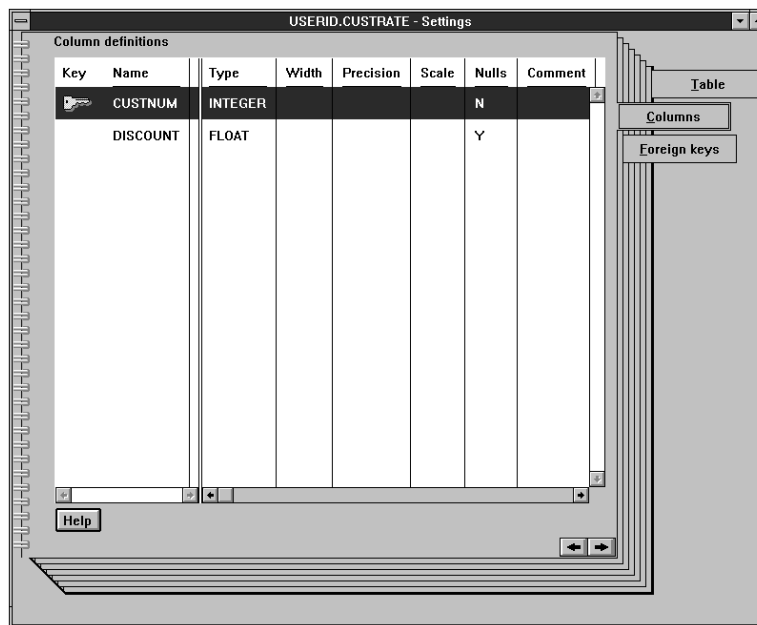


Figure 161. Table Settings notebook - Columns page

The **Columns** page of a table or view's **Settings** notebook contains the following table/view definition information:

- **Key:** a yellow right-pointing key icon indicates a primary key. A gray left-pointing key indicates a foreign key.
- **Name:** the name of the table/view column.
- **Type:** the database data type of the column.
- **Width:** the number of characters in a character-type column.

## Using the Data Access Builder

- **Precision:** the total number of digits in a decimal-type column.
- **Scale:** the number of digits in the fractional part of the number.
- **Nulls:** refers to whether the column can contain a null value (is nullable). A null value means that no value is set for the column. Key columns cannot be nullable.

### Foreign Keys Page

Clicking on the **Foreign keys** tab displays the **Foreign keys** page (see Figure 162).

**Note:** Views do not have a **Foreign keys** page in their **Settings** notebook.

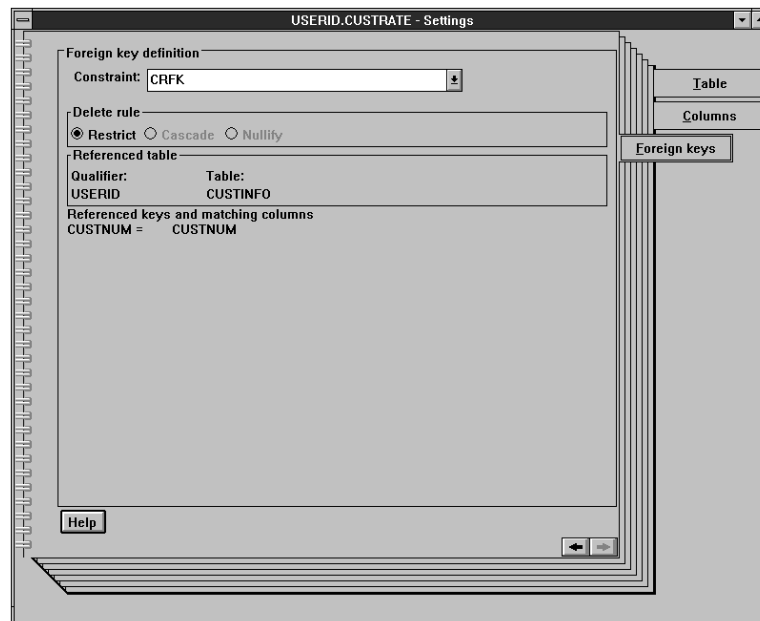


Figure 162. Table Settings notebook - Foreign keys page'.

Click on the down arrow in the **Constraint** text field to see a list of all constraints that apply to the table. When you select a constraint from the list, the **Foreign keys** page is updated with the following information:

- Constraints on the table
- The delete rule for each constraint
- The referenced table for each constraint
- Referenced keys and matching columns for each constraint



## Using the Data Access Builder

### Changes in the Database Table/View Definition

Changes made to the database table or view definition are not automatically reflected in the **Settings** notebook. Synchronization between the database table and the table object you created with Data Access Builder is not maintained. The only connection between the actual table and the Data Access Builder object occurs once - when the object is first created.

If the changes to the definition of the database table or view are substantial, your generated Data Access Builder code may no longer be usable. However, some updates to a table definition may not affect your generated code's viability, for example:

- Adding a column
- Deleting a column that was unmapped in your class object
- Adding a foreign key

To recover from a change in the database definition, you must delete the changed table (which also deletes all classes mapped to it), and recreate the mapping (refer to sections: “Deleting Class and Table/View Objects” on page 567 and Chapter 49, “Creating Table and Class Objects” on page 559).

## **Using the Data Access Builder**

## Chapter 50. Generating the Source Code

This chapter explains how to complete the mapping from a table (or view) to a class by optionally customizing the default settings, and then how to generate the source code.

Other topics also covered in this chapter include:

- Viewing the generated files
- Saving your Data Access Builder session
- Exiting the Data Access Builder tool
- What to do once you have finished generating your classes

---

### Customizing the Class Mapping

Each class object has a **Settings** notebook that allows you to make changes to a number of the class object's default settings. To open the class object's **Settings** notebook, click mouse button two on the class object icon and select **Open settings** (Alternatively, you can simply double-click on the class object icon to open its **settings** notebook.)

### Class Settings Notebook - Names Page

The class object's **Settings** notebook opens on the **Names** page (see Figure 163).

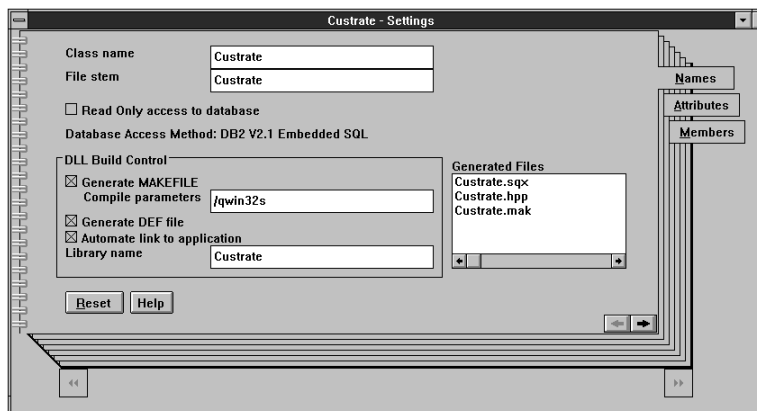


Figure 163. Class Settings notebook - Names page

## Using the Data Access Builder

- **Class name**

This page allows you to change the name of the generated class by editing the **Class name** entry field. Changes to this field will also be reflected in the class object icon displayed in the client area.

- **File stem**

The generated filenames can also be changed by editing the **File stem** field. The generated filenames consist of the specified **File stem**, and a file extension, which is generated by Data Access Builder and cannot be changed.

Data Access Builder uses specific file extensions so that the files it generates can be used by the appropriate tools. For example, .VBE indicates an export file created by Data Access Builder and used by Visual Builder to import necessary information about the generated classes.

**Note:** If your file system limits the length of filenames, Data Access Builder truncates the **File stem** accordingly.

- **Read Only**

The **Read Only** checkbox allows you to specify that all instances of your class not be permitted to write, update, or delete your database information. An exception will be thrown if a write access is attempted with classes that have been created with this option selected.

**Note:** Data Access Builder may place a check in this checkbox if it can determine that a table or view is read only.

- **Generate a makefile**

You can also mark the **Generate a makefile** checkbox if you want Data Access Builder to create a makefile. This option is particularly useful if you're not using WorkFrame to build your application and would otherwise have to create a makefile yourself.

- **Generated Files**

The listbox entitled **Generated Files** lists all the filenames for the files that will be generated. The files will be named as specified by your selections on this page.

## Class Settings Notebook - Attributes Page

Clicking on the **Attributes** tab displays the **Attributes** page (see Figure 164 on page 575). The left side of the list displays the table or view information; the right side displays the class mapping.

## Using the Data Access Builder

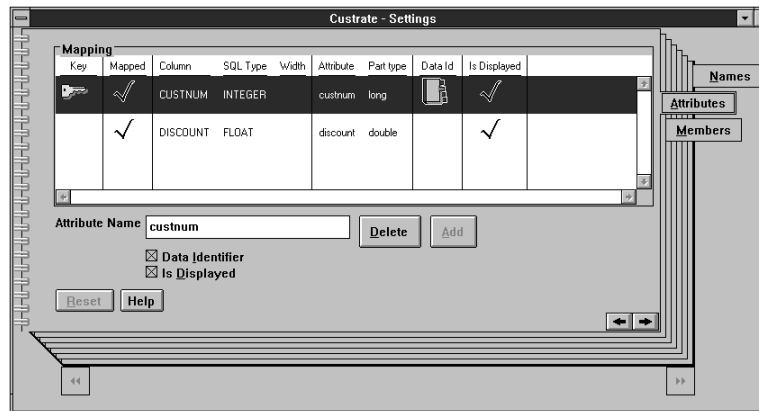


Figure 164. Class Settings notebook - Attributes page

Before generating the source code for your classes, you may want to customize the mappings to suit your needs. By default, each column in the table is automatically mapped to an attribute in the class object. However, you may not require access to every column in the table and will therefore want to have column-to-attribute mappings only for the columns you will use.

- **Delete**

Data Access Builder allows you to delete any mappings you do not require, and add them again if you change your mind or make a mistake.

To delete a mapping, simply select the mapping you want to remove from the list and click on the **Delete** push button.

The **Attribute** column of the list is updated with **<none>** and the remaining columns of the class mapping information are cleared.

**Note:** If you delete a mapping with the only (remaining) data identifier, you will not be allowed to exit the notebook until you assign a data identifier to an attribute. All class objects must contain at least one data identifier (see “Data Identifiers and Primary Keys” on page 576).

- **Add**

To add a mapping, select a deleted mapping from the list and click on the **Add** push button. The class mapping information is restored to its default values.

**Note:** You cannot use the add function to add a mapping that has not been previously deleted (for example, to compensate for a change in the table definition - see “Changes in the Database Table/View Definition” on page 571).

## Using the Data Access Builder

- **Attribute Name**

The name of a mapped attribute can also be changed. To change the attribute name, select the mapping you want to change, edit the **Attribute Name** entry field, and press **Enter**.

- **Is Displayed**

The **Is Displayed** checkbox allows you specify whether the selected attribute is included as part of the returned value from the `forDisplay` method. `forDisplay` returns an `IString` containing all the class attributes that have their **Is Displayed** checkbox checkmarked, in the format specified by the `asString` method.

- **Data Identifier**

The **Data Identifier** checkbox allows you to specify whether an attribute is to be designated as part of the `DataId` for the class. The `DataId` can consist of one or more attributes; however, each class must have at least one data identifier. See “Data Identifiers and Primary Keys” for more information.

### ***Data Identifiers and Primary Keys***

In the key column of the table/view information, the primary key for the table or view is indicated by one or more gold (right-pointing) keys. A grey (left-pointing) key indicates a foreign key. By default, a table or view's primary key is mapped as the class object's **DataId**; therefore, the gold key is also indicated on the **Attributes** page as the *DataId*.

Database Management Systems (DBMS) use primary keys to uniquely identify a row. If a row cannot be uniquely identified, then an update, retrieve, or delete operation may affect more than just the desired row.

Class objects use data identifiers to identify a row uniquely. Although some database products do not require that a table or view has a primary key, a class object must have a data identifier and the value of the data identifier cannot be nullable (cannot be null) and must be unique.

If the table does not have a primary key, the first attribute is selected as the default data identifier (this default can be changed, see “Class Settings Notebook - Attributes Page” on page 574). Your application *must* ensure this attribute contains unique values. If values kept in the data identifier identify more than one row, errors occur for the retrieve operation and multiple rows are affected for update and delete operations.

## Using the Data Access Builder

### Class Settings Notebook - Members Page

The **Members** section displays the class information on a series of pages. Clicking on the **Members** tab displays the **Summary** page of the **Members** section (see Figure 165).

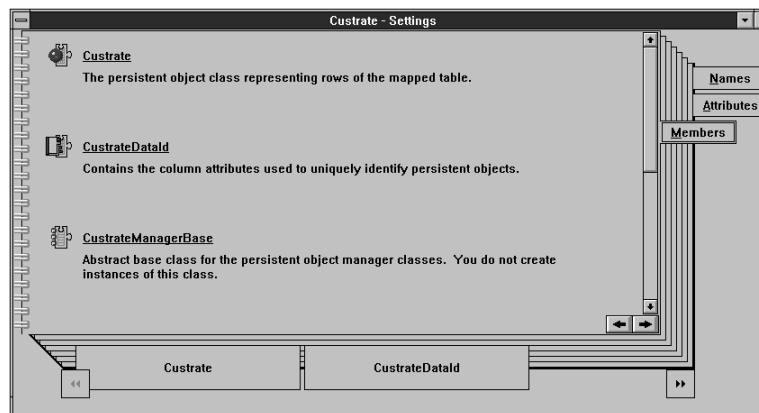


Figure 165. Class Settings notebook - Members section (Summary page)

The **Summary** page displays a short description of each of the generated classes. For Visual Builder classes, the part icon that is imported into the Visual Builder palette is displayed on the **summary** page next to the class description.

Each generated class is described in more detail in the subsequent pages (one class is described per page). For each generated class, two list boxes are displayed on its page. In the top list box, the class attributes are displayed; the bottom list box displays the class methods/actions. Figure 166 on page 578, shows a page displaying the details for the generated data-object class.

## Using the Data Access Builder

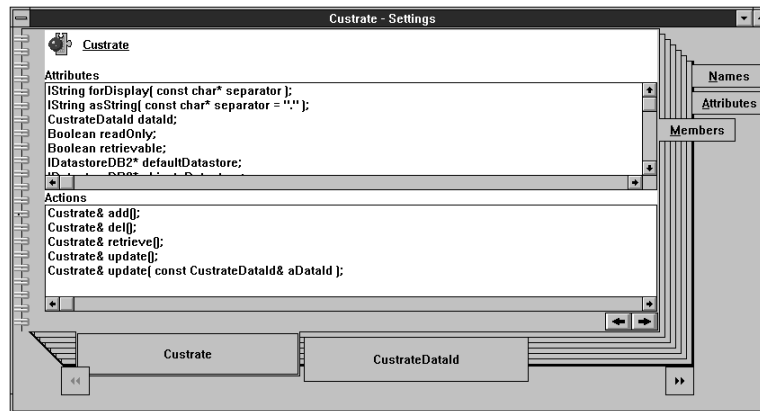


Figure 166. Class Settings notebook - Members section (Data Object page)

### Navigating the Pages of the Members Section

To navigate the pages of the **Members** section:

- Click on a tab displayed at the bottom of the notebook to display that page.
- The arrows near the bottom right corner of the notebook are used to advance to the next/previous page of the notebook.
- The small arrows (to the left and right of the tabs at the bottom of the notebook) are used to scroll into view any tabs not currently visible.
- At any time, clicking on the **Members** tab on the side of the notebook, brings you back to the **Summary** page.

---

## Generating the Class Source Code

After customizing the table/view to class-object mapping (if required) you are ready to generate the source code. Note however, that a class must have at least one attribute mapped to a column, and at least one data identifier defined, before you can generate code for it.

To generate source code for a class object, click mouse button two on the class object icon and select **Generate** from the pop-up menu. The source code is generated according to the previously selected generation options and any customizations made to the class object notebook settings.

**Note:** You cannot choose the code type through this menu item; to change the generated code type, you must create a new class from the table/view object's pop-up menu (refer to “Generating Different Kinds of Classes for the Same Table/View” on page 566).



## Using the Data Access Builder

After source code is generated, the class icon color changes to one of the following:

- Yellow for Visual Builder
- Green for IDL
- Purple for C++

If you open the class object's **Settings** notebook and make any changes to the **Names** or **Attributes** page, the class object icon color will change back to black-and-white, signifying that code has not been generated for the new configuration. However, the generated files for the old configuration are not erased, but they will be overwritten if the **File stem** is not changed in the new configuration and code is regenerated.

By default, the source files are saved in your project's target directory if you are using Workframe; otherwise, they are saved in the current directory. To change the default directory, edit the **Files will be generated in** entry field in the **Generate Options** dialog box (see “Specifying Default Generation Options” on page 563).

**Note:** Once the source code has been generated, the generated files should not be modified. Any manual changes to these files will be lost the next time the source code is generated (that is, if it is re-generated).

---

### Saving a Data Access Builder Session

At any time, you can save your Data Access Builder session. All information about the session is saved.

1. Select **Save** from the **File** menu. If this is the first time you are saving this session, the **Save as** window displays.
2. Type the name you want to assign to the session in the **Filename** entry field. It is recommended that the file extension be **.DAX**.
3. Indicate the directory you want the information stored in by selecting one of the drives in the **Drives** list box.
4. Click on **OK** to save the session.

The session is saved in the name shown in the title bar.

On subsequent saves of the same session, the **Save** window will not appear.

### Saving a Data Access Builder Session under Another Name

At any time, you can save the current Data Access Builder session under a different name.

1. Choose **Save as** from the **File** menu.
2. Type the name you want to assign to this session in the **Filename** entry field. It is recommended that the file extension be **.DAX**.

## Using the Data Access Builder

3. Indicate the directory you want the information stored in by selecting one of the drives in the **Drives** list box.
4. Choose **OK** to save the new session name.

---

## Viewing Generated Files

Data Access Builder allows you to view the generated files without leaving the tool. You may be particularly interested in viewing the .HPP file for the generated data access class definitions. (While browsing this code, you may want to refer to the *Open Class Library Reference* and *Open Class Library User's Guide* for detailed information on the classes.)

Once classes are generated, the **View source...** menu option is added to the class icon's pop-up menu. To view the generated files:

1. Click mouse button two on the class object icon to display the pop-up menu, and select **View source...**

The **View Source** dialog box appears.

2. Select the files you would like to view from the list box and click on the **View** push button.

Data Access Builder launches a WorkFrame view tool session for each file you selected.

3. When you are finished browsing the files, close each session and return to the Data Access Builder tool.

---

## Using the Generated Files

After completing all the steps above, you are now ready to use the generated files. Please refer to the "Data Access Builder" section of the *Open Class Library User's Guide*.



## Chapter 51. Getting Started with ODBC

This chapter explains how to use ODBC drivers to access your database. Topics covered in this chapter include:

- About ODBC drivers
- Installing the ODBC drivers
- Error messages
- Windows 3.x run-time support
- A general discussion of isolation levels and locking
- The structure of ODBC.INI

---

### About ODBC Drivers

The ODBC drivers are compliant with the Microsoft Open Database Connectivity (ODBC 2.0) specification. ODBC is a specification for an application program interface (API) that enables applications to access multiple database management systems using Structured Query Language (SQL) as a base.

ODBC permits maximum interoperability - a single application can access many different database management systems. This enables an ODBC developer to develop, compile, and ship an application without targeting a specific type of database management system (DBMS). Users can then add the database drivers, which link the application to the DBMS of their choice.

### ODBC Supported Database Systems

Access to a multitude of database systems is available through the ODBC driver manager and related drivers. With this release of VisualAge for C++, *official support* is only extended to ODBC CLI access to Sybase System 10, Oracle 7, and DB2 through their respective ODBC drivers.

**Note:** Although any database product supported by a compatible ODBC 2.0 driver can be accessed reliably through Data Access Builder's ODBC CLI support, ***only the ODBC drivers for the aforementioned officially-supported database products can be used in the development of applications for which IBM Service is requested.***

For DB2, the ODBC driver that comes with the **IBM Database 2** product is the officially supported driver. For all other officially supported database products, the INTERSOLV DataDirect ODBC drivers supplied with the IBM VisualAge for C++ for Windows product are the officially supported drivers.

## Using the Data Access Builder

---

### Installing the ODBC Drivers

The ODBC drivers were installed when you first installed the IBM VisualAge for C++ for Windows product, or in the case of DB2, when you installed the DB2 product. If you did not install any ODBC drivers, or if you wish to install additional drivers, you can run the installation program again and selectively install the desired ODBC drivers.

#### Driver names

The filenames for all the ODBC drivers that come with VisualAge for C++ contain the prefix "IB". The file extension is .DLL. This indicates that they are dynamic link libraries. For example, the Oracle 7 driver filename is IBOR7nn.DLL, where *nn* is the revision number of the driver.

### Configuring Data Sources

A data source consists of a DBMS and any remote operating system and network necessary to access it. After the drivers have been installed, the data source must be configured using the ODBC Administrator program, which is located in the Windows Control Panel. To start the ODBC administrator, double-click the ODBC icon.

Because Windows 95 and Windows NT can host multiple users, each user must configure their own data sources.

For detailed configuration information for the specific driver you wish to configure, refer to the appropriate section of the chapter Chapter 52, "Supported ODBC Drivers" on page 587.

---

### Windows 3.x Run-time Support

For users creating applications which will run in the Windows 3.x environment, 16-bit ODBC drivers are supplied. For more information on creating applications for Windows 3.1 using VisualAge for C++ and Microsoft's WIN32S library, refer to the *Programming Guide* and the VisualAge for C++ README file.

---

### Error Messages

Error messages may come from:

- An ODBC driver
- The data source
- The ODBC driver manager

**Note:** When using the Data Access Builder's class library, error messages are reported in the error text of the appropriate exception class. For more

## Using the Data Access Builder

information, refer to the Data Access Builder section of the *Open Class Library Reference*.

An error reported on an ODBC driver has the following format:

[vendor] [ODBC\_component] message

*ODBC\_component* is the component in which the error occurred. For example, an error message from INTERSOLV's Sybase System 10 driver would look like this:

[INTERSOLV] [ODBC Sybase System 10 driver] Invalid precision specified.

If you get this type of error, check the last ODBC call your application made for possible problems.

An error that occurs in the data source includes the data source name, in the following format:

[vendor] [ODBC\_component] [data\_source] message

With this type of message, *ODBC\_component* is the component that received the error from the data source indicated. For example, you may get the following message from an Oracle data source:

[INTERSOLV] [ODBC Oracle driver] [Oracle] ORA-0919: specified length too long for CHAR column

If you get this type of error, you did something incorrectly with the database system. Check your database system documentation for more information or consult your database administrator. In this example, you would check your Oracle documentation.

The driver manager is a DLL that establishes connections with drivers, submits requests to drivers, and returns results to applications. An error that occurs in the driver manager has the following format:

[vendor] [ODBC DLL] message

*vendor* is the ODBC driver manager vendor. For example, an error from the Microsoft driver manager might look like this:

[Microsoft] [ODBC DLL] Driver does not support this function

If you get this type of error, consult the *Programmer's Reference* for the Microsoft ODBC Software Development Kit available from Microsoft.

---

### Locking and Isolation Levels

This section discusses locking and isolation levels and how their settings can affect the data you retrieve. Different database systems support different locking and isolation levels. See the section "Isolation and Lock Levels Supported" in the appropriate driver section.

#### Locking

Locking is a database operation that restricts a user from accessing a table or record. Locking is used in situations where more than one user might try to use the same table or record at the same time. By locking the table or record, the system ensures that only one user at a time can affect the data.

Locking is a vital activity in multiuser databases, where different users can try to access or modify the same records concurrently. While such concurrent database activity is desirable, it can create problems. Without locking, for example, if two users try to modify the same record at the same time, they might encounter problems ranging from retrieving bad data to deleting data that the other user needs. However, if the first user to access a record can lock that record to temporarily prevent other users from modifying it, such problems can be avoided. Locking provides a way to manage concurrent database access while minimizing the various problems it can cause.

#### Isolation Levels

An isolation level represents a particular locking strategy employed in the database system to improve data consistency. The higher the isolation level, the more complex the locking strategy behind it. The isolation level provided by the database determines whether a transaction will encounter the following behaviors in data consistency:

- **Dirty reads**

User 1 modifies a row. User 2 reads the same row before User 1 commits. User 1 performs a rollback. User 2 has read a row that has never really existed in the database. User 2 may base decisions on false data.

- **Non-repeatable reads**

User 1 reads a row but does not commit. User 2 modifies or deletes the same row and then commits. User 1 rereads the row and finds it has changed (or has been deleted).

- **Phantom reads**

User 1 uses a search condition to read a set of rows but does not commit. User 2 inserts one or more rows that satisfy this search condition, then commits. User 1 rereads the rows using the search condition and discovers rows that were not present before.

## Using the Data Access Builder

Isolation levels represent the database system's ability to prevent these behaviors. The American National Standards Institute (ANSI) defines four isolation levels:

- Read uncommitted (0)
- Read committed (1)
- Repeatable read (2)
- Serializable (3)

In ascending order (0-3), these isolation levels provide an increasing amount of data consistency to the transaction. At the lowest level, all three behaviors can occur. At the highest level, none can occur. The success of each level in preventing these behaviors is due to the locking strategies that they employ, which are as follows:

- **Read uncommitted (0)**

Locks are obtained on modifications to the database and held until end of transaction (EOT). Reading from the database does not involve any locking.

- **Read committed (1)**

Locks are acquired for reading and modifying the database. Locks are released after reading but locks on modified objects are held until EOT.

- **Repeatable read (2)**

Locks are obtained for reading and modifying the database. Locks on all modified objects are held until EOT. Locks obtained for reading data are held until EOT. Locks on non-modified access structures (indexes, hashing structures, etc.) are released after reading.

- **Serializable (3)**

All data read or modified is locked until EOT. All access structures that are modified are locked until EOT. Access structures used by the query are locked until EOT.

Figure 167 shows what data consistency behaviors can occur at each isolation level.

Figure 167. Isolation Levels and Data Consistency

Level	Dirty Read	Nonrepeatable Read	Phantom Read
0, Read uncommitted	Yes	Yes	Yes
1, Read committed	No	Yes	Yes
2, Repeatable read	No	No	Yes
3, Serializable	No	No	No

## Using the Data Access Builder

While higher isolation levels provide better data consistency, this consistency can be costly in terms of the concurrency provided to individual users. Concurrency is the ability of multiple users to access and modify data simultaneously. As isolation levels increase, so does the chance that the locking strategy used will create problems in concurrency.

*Put another way:* the higher the isolation level, the more locking involved, and the more time users may spend waiting for data to be freed by another user. Because of this inverse relationship between isolation levels and concurrency, you must consider how people use the database before choosing an isolation level. You must weigh the trade-offs between data consistency and concurrency, and decide which is more important.





## Chapter 52. Supported ODBC Drivers

This chapter contains a section for each supported database driver. Each driver's section is structured so that the information you need to reference first is at the beginning of the section. First, it lists system requirements for your operating environment. Next, it explains how to configure a data source. Finally, it explains how to connect to that data source.

### Usage Note

The sections which explain how to connect to a data source contain reference information which is to be used in conjunction with the Data Access Builder class library information (refer to the *Open Class Library User's Guide* and the *Open Class Library Reference*).

### Notes:

1. The section "Configuring Data Sources" in each subsequent driver section instructs you to start the ODBC Administrator. To start the ODBC Administrator, double-click the ODBC icon in the Windows Control Panel.
2. On Windows 95 and Windows NT systems, the ODBC drivers are 32-bit drivers. All required network software supplied by your database system vendors must be 32-bit compliant. Consult the "System Requirements" section for specific requirements for each database driver.
3. On Microsoft Windows 3.x systems, the ODBC drivers are 16-bit drivers. All required network software supplied by your database system vendors must be 16-bit compliant. Consult the "System Requirements" section of the appropriate driver section for specific requirements for each ODBC driver.

---

## DB2 Driver

For DB2, the ODBC driver that comes with the **IBM Database 2** product is the officially supported driver. For information on configuring and using this driver, refer to the sections:

- Setting Up ODBC Applications
- Configuring ODBC.INI

in the chapter "Configuring CLI and Running Sample Applications" in the *DB2 Call Level Interface Guide and Reference*.

## Using the Data Access Builder

---

### Oracle 7 Driver

The Oracle 7 driver is supported in the Windows, Windows 95 and Windows NT environments.

### System Requirements

The Oracle SQL\*Net product is required to access remote Oracle databases.

### Configuring Data Sources

To configure an Oracle data source:

1. Start the ODBC Administrator. A list of data sources appears.
2. If you are configuring a new data source, click **Add**. A list of installed drivers appears. Select the Oracle 7 driver and click **OK**.

If you are configuring an existing data source, select the data source name and click **Setup**.

The setup dialog box appears.

3. Specify values as follows:

**Data Source Name:** A string that identifies this Oracle data source configuration in ODBC.INI. Examples include "Accounting" or "Oracle- Serv1."

**Description:** An optional long description of a data source name. For example, "My Accounting Database" or "Oracle on Server number 1."

**Server Name:** The SQL\*Net connection string designating the server and database to be accessed. The information required varies depending on the SQL\*Net driver you are using. The section "Connecting to a Data Source Using a Connection String" on page 589 gives the format of the SQL\*Net connection string.

The following values are optional:

**Server List:** The list of SQL\*Net connection strings that will appear in the logon dialog box. Separate the strings with commas. If the SQL\*Net connection string contains a comma, enclose it in quotation marks; for example, "Serv,1", "Serv,2", "Serv,3."

**Default User Name:** The default user name used to connect to your Oracle database. A default user name is required only if security is enabled on your database. Your ODBC application may override this value or you may override this value in the logon dialog box or connection string.

**Lock Time Out:** A value of 0 or -1 that specifies whether Oracle should wait for a lock to be freed before raising an error when processing a Select...For Update statement. Values can be -1 (wait forever) or 0 (don't wait). The default is -1.

## Using the Data Access Builder

**Array Size:** The number of bytes the driver uses for fetching multiple rows. Values can be an integer from 0 to 65536; the default is 60000. Larger values increase throughput by reducing the number of times the driver fetches data across the network. Smaller values increase response time, as there is less of a delay waiting for the server to transmit data.

**Catalog Comments:** Check this box if you want to retrieve the contents of the COMMENTS column for catalog functions. Retrieving the COMMENTS column may reduce the performance of your data catalog operations.

4. Click **Translate** to perform a translation of your data from one character set to another. The Select Translator dialog box appears in which you can select a translator to perform the translation. Click **OK** to leave this dialog box and perform the translation.

The translators that are listed in this dialog box are determined by the values listed in the ODBC Translators section of your `odbcinst.ini` file.

5. Click **OK** to write these values to `ODBC.INI`. These values are now the defaults when you connect to the data source. You can change the defaults by configuring your data source again. You can override the defaults by connecting to the data source using a connection string with alternate values.

### Connecting to a Data Source Using a Logon Dialog Box

Some ODBC applications display a logon dialog box when you are connecting to a data source. In these cases, the data source name has already been specified. In this dialog box, do the following:

1. Type the SQL\*Net connection string of the computer containing the Oracle database tables you want to access or select the string from the Server Name drop-down list, which displays the names you specified in the setup dialog box.
2. If required, type your Oracle user name.
3. If required, type your Oracle password.
4. Click **OK** to log on to the Oracle database installed on the server you specified and to update the values in `ODBC.INI`.

### Connecting to a Data Source Using a Connection String

If your application requires a connection string to connect to a data source, you must specify the data source name that tells the driver which `ODBC.INI` section to use for the default connection information. Optionally, you may specify *attribute=value* pairs in the connection string to override the default values stored in `ODBC.INI`. These values are not written to `ODBC.INI`.

You can specify either long or short names in the connection string. The connection string has the form:

## Using the Data Access Builder

`DSN=data_source_name[:attribute=value [:attribute=value]...]`

An example of a connection string for Oracle is

`DSN=Accounting;SRVR=X:QESRVR;UID=JOHN;PWD=XYZZY`

If the server name contains a semicolon, enclose it in quotation marks:

`DSN=Accounting;SRVR="X:QE;SRVR";UID=JOHN;PWD=XYZZY`

**Note:** When using the Data Access Class Library's `IDatastoreODBC.connect()` method, the data source name (DSN), userid (UID), and password (PWD) attributes are passed as separate parameters from the rest of the connection string. Using the above connection string as an example, the equivalent `connect()` method call would be:

```
connect("Accounting", "SRVR='X:QE;SRVR'", "JOHN", "XYZZY");
```

Figure 168 gives the long and short names for each attribute, as well as a description.

The defaults listed in the table are initial defaults that apply when no value is specified in either the connection string or in the data source definition in `ODBC.INI`. If you specified a value for the attribute when configuring the data source, that value is your default.

Figure 168 (Page 1 of 2). Oracle Connection String Attributes

Attribute	Description
DataSourceName (DSN)	A string that identifies an Oracle data source configuration in <code>ODBC.INI</code> . Examples include "Accounting" or "Oracle-Serv1."
LogonID (UID)	The logon ID (user name) that the application uses to connect to your Oracle database. A logon ID is required only if security is enabled on your database. If so, contact your system administrator to get your logon ID.
Password (PWD)	Your password.
LockTimeOut (LTO)	A value that specifies whether Oracle should wait for a lock to be freed before raising an error when processing a <code>Select...For Update</code> statement. Values can be -1 (wait forever, the initial default) or 0 (don't wait).
ArraySize (AS)	The number of bytes the driver uses for fetching multiple rows. Values can be an integer from 0 to 65536. The initial default is 60000. Larger values increase throughput by reducing the number of times the driver fetches data across the network. Smaller values increase response time, as there is less of a delay waiting for the server to transmit data.

## Using the Data Access Builder

Figure 168 (Page 2 of 2). Oracle Connection String Attributes

Attribute	Description
ServerName (SRVR)	<p>The SQL*Net connection string designating the server and database to be accessed. The information required varies depending on the SQL*Net driver you are using.</p> <p>For remote servers, the SQL*Net connection string has the following form:</p> <p><i>driver_prefix:computer_name[: sid]</i></p> <p><i>driver_prefix</i> identifies the network protocol being used. The driver prefix can be as follows: P (named pipes), X (SPX), B (NetBIOS), T (TCP/IP), D (DECNet), A (Oracle Async), or TNS (SQL*net 2.0). Check your Oracle documentation for other protocols.</p> <p><i>computer_name</i> is the name of the Oracle Listener on your network.</p> <p><i>sid</i> is the Oracle System Identifier and refers to the instance of Oracle running on the host. This item is required when connecting to systems that support more than one instance of an Oracle database.</p> <p>For local servers, the SQL*Net connection string has the form:</p> <p><i>database_name</i></p> <p><i>database_name</i> identifies your Oracle database.</p> <p>If the SQL*Net connection string contains semicolons, enclose it in quotation marks. See your SQL*Net documentation for more information.</p>
CatalogComments (CC)	<p>CatalogComments={0 1}. This attribute specifies whether the driver returns the contents of the COMMENTS column for catalog functions. CatalogComments=1 returns COMMENTS. Retrieving the COMMENTS column may reduce the performance of your data catalog operations. CatalogComments=0 does not return COMMENTS (the initial default).</p>

## Oracle 7 Data Types

Figure 169 on page 592 shows how the Oracle 7 data types are mapped to the standard ODBC data types.

## Using the Data Access Builder

Figure 169. Oracle 7 Data Types

Oracle 7	ODBC
Char	SQL_CHAR
Date	SQL_TIMESTAMP
Long	SQL_LONGVARCHAR
Long Raw	SQL_LONGVARBINARY
Number	SQL_DOUBLE
Number(p,s)	SQL_DECIMAL
Raw	SQL_VARBINARY
Varchar2	SQL_VARCHAR

## Isolation and Lock Levels Supported

Oracle 7 supports isolation level 2 (repeatable read) only. Both versions support record-level locking. See “Locking and Isolation Levels” on page 584 for a discussion of these topics.

## ODBC Implementation Level

The Oracle 7 driver supports the Core, Level 1, and the following Level 2 functions:

- SQLDataSources
- SQLExtendedFetch(forward scrolling only)
- SQLMoreResults
- SQLNativeSql
- SQLNumParams
- SQLSetScrollOptions
- SQLBrowseConnect
- SQLProcedures
- SQLPrimaryKeys

The driver also supports the core SQL grammar.

## Number of Connections and Statements Supported

Oracle 7 supports multiple connections and multiple statements per connection.

---

## Sybase System 10 Driver

The Sybase System 10 driver supports the SQL Server System 10 database system from Sybase in the Windows, Windows 95 and Windows NT environments.

### System Requirements

You must install the Sybase Open Client-Library and the appropriate Sybase Net-Library to gain access to System 10.

### Configuring Data Sources

To configure a System 10 data source:

1. Start the ODBC Administrator. A list of data sources appears.
2. If you are configuring a new data source, click **Add**. A list of installed drivers appears. Select the Sybase System 10 driver and click **OK**.

If you are configuring an existing data source, select the data source name and click **Setup**.

The setup dialog box appears.

3. Specify values as follows:

**Data Source Name:** A string that identifies this Sybase System 10 data source configuration in ODBC.INI. Examples include "Accounting" or "Sys10-Serv1."

**Description:** An optional long description of a data source name. For example, "My Accounting Database" or "System 10 on Server number 1."

**Server Name:** The name of the server that contains the System 10 tables you want to access. If not supplied, the server name in the DSQUERY environment variable is used.

The following values are optional:

**Server List:** The list of servers that appear in the logon dialog box. Separate the server names with commas.

**Database Name:** The name of the database to which you want to connect by default. If you do not specify a value, the default is the database defined by the system administrator for each user.

**Database List:** The databases that appear in the logon dialog box. Separate the names with commas.

**Default Logon ID:** The default logon ID used to connect to your System 10 database. This ID is case-sensitive. A logon ID is required only if security is enabled for the database you are connecting to. Your ODBC application may override this value or you may override this value in the logon dialog box or connection string.

## Using the Data Access Builder

**Interfaces File:** The pathname of the interfaces file. The default is the normal Sybase interfaces file.

**Password Encryption:** A value of 0 or 1 that determines whether password encryption can be performed from the Open Client Library to the server. A value of 1 enables this password encryption; a value of 0, the default, does not.

**Charset:** The name of a character set corresponding to a subdirectory in \$SYBASE/charsets. The default is the setting on the System 10 server.

**Workstation ID:** The workstation ID used by the client.

**Language:** The national language corresponding to a subdirectory in \$SYBASE/locales. The default is English.

**Application Name:** The name used by System 10 to identify your application.

**Yield Proc:** An integer value of 0, 1, or 3 that determines whether you can work in other applications when Sybase System 10 is busy. This attribute is useful to users of ODBC applications.

- 0 (peek and dispatch), which causes the driver to check the Windows message queue and send any messages to the appropriate Windows application.
- 1 (no yielding, the default), which does not let you work in other applications.
- 3 (dispatch via Windows Yield function), which turns control over to the Windows kernel. The Windows kernel checks the message queue and sends any messages to the appropriate application window.

It is recommended that you use the value 1.

The following are performance settings:

**Optimize Prepare:** A value of 0, 1, or 2 that determines whether stored procedures are created on the server for every call to SQLPrepare.

When set to 0, stored procedures are created for every call to SQLPrepare. This setting can result in bad performance.

When set to 1, the initial default, the driver creates stored procedures only if the statement contains parameters. Otherwise, the statement is cached and executed directly at SQLExecute time.

When set to 2, the driver never creates stored procedures.

**Array Size:** The number of rows the driver retrieves when fetching from the server. This is not the number of rows given to the user. The default is 10 rows.

**Select Method:** A value of 0 or 1 that determines whether database cursors are used for Select statements. When set to 0, the default, database cursors are used;



## Using the Data Access Builder

when set to 1, Select statements are executed directly without using database cursors. A setting of 1 limits the data source to one active statement and one active connection.

**Packet Size:** A value of -1, 0, or  $x$  that determines the number of bytes per network packet transferred from the database server to the client. The correct setting of this attribute can improve performance.

When set to 0, the default, the driver uses the default packet size as specified in the System 10 server configuration.

When set to -1, the driver computes the maximum allowable packet size on the first connect to the data source and saves the value in the `odbc.ini` file.

When set to  $x$ , an integer from 1 to 10, which indicates a multiple of 512 bytes (for example, 6 means to set the packet size to  $6 * 512 = 3072$  bytes).

For you to take advantage of this connection attribute, you must configure the System 10 server for a maximum network packet size greater than or equal to the value you specified for `PacketSize`. For example,

```
sp_configure "maximum network packet size", 5120
reconfigure
Restart System 10 Server
```

Note that the ODBC specification identifies a connect option, `SQL_PACKET_SIZE`, that offers this same functionality. To avoid conflicts with applications that may set both the connection string attribute and the ODBC connect option, they have been defined as mutually exclusive. If `PacketSize` is specified, you will receive a message "Driver Not Capable" if you attempt to call `SQL_PACKET_SIZE`. If you do not set `PacketSize`, then application calls to `SQL_PACKET_SIZE` are accepted by the driver.

4. Click **Translate** to perform a translation of your data from one character set to another. The Select Translator dialog box appears in which you can select a translator to perform the translation. Click **OK** to leave this dialog box and perform the translation.

The translators that are listed in this dialog box are determined by the values listed in the ODBC Translators section of your `odbcinst.ini` file.

5. Click **OK** to write these values to `ODBC.INI`. These values are now the defaults when you connect to the data source. You can change the defaults by configuring your data source again. You can override the defaults by connecting to the data source using a connection string with alternate values.

## Using the Data Access Builder

### Connecting to a Data Source Using a Logon Dialog Box

Some ODBC applications display a Logon dialog box when you are connecting to a data source. In these cases, the data source name has already been specified. In this dialog box, do the following:

1. Type the case-sensitive name of the server containing the System 10 database tables you want to access or select the name from the Server Name drop-down list, which displays the server names you specified in the setup dialog box.
2. If required, type your case-sensitive login ID.
3. If required, type your case-sensitive password for the system.
4. Type the name of the database you want to access (case-sensitive) or select the name from the Database drop-down list, which displays the names you specified in the setup dialog box.
5. Click **OK** to complete the logon and to update the values in ODBC.INI.

### Connecting to a Data Source Using a Connection String

If your application requires a connection string to connect to a data source, you must specify the data source name that tells the driver which ODBC.INI section to use for the default connection information. Optionally, you may specify *attribute=value* pairs in the connection string to override the default values stored in ODBC.INI. These values are not written to ODBC.INI.

You can specify either long or short names in the connection string. The connection string has the form:

`DSN=data_source_name[:attribute=value [:attribute=value]...]`

An example of a connection string for Sybase System 10 is

`DSN=SYS10 TABLES;SRVR=QESRVR;DB=PAYROLL;UID=JOHN;PWD=XYZZY`

**Note:** When using the Data Access Class Library's `IDatastoreODBC connect()` method, the data source name (DSN), userid (UID), and password (PWD) attributes are passed as separate parameters from the rest of the connection string. Using the above connection string as an example, the equivalent `connect()` method call would be:

`connect("SYS10 TABLES", "SRVR=QESRVR;DB=PAYROLL", "JOHN", "XYZZY");`

Figure 170 on page 597 gives the long and short names for each attribute, as well as a description.

The defaults listed in the table are initial defaults that apply when no value is specified in either the connection string or in the data source definition in ODBC.INI.

## Using the Data Access Builder

If you specified a value for the attribute when configuring the data source, that value is your default.

*Figure 170 (Page 1 of 3). Sybase System 10 Connection String Attributes*

Attribute	Description
DataSourceName (DSN)	A string that identifies a single connection to a System 10 database. Examples include "Accounting" or "Sys10-Serv1."
ServerName (SRVR)	The name of the server containing the System 10 tables you want to access. If not supplied, the initial default is the server name in the DSQUERY environment variable.
LogonID (UID)	The default logon ID used to connect to your System 10 database. This ID is case-sensitive. A logon ID is required only if security is enabled on your database. If so, contact your system administrator to get your logon ID.
Password (PWD)	A case-sensitive password.
Database (DB)	The name of the database to which you want to connect.
Language (LANG)	The national language corresponding to a subdirectory in \$SYBASE/locales.
Charset (CS)	The name of a character set corresponding to a subdirectory in \$SYBASE/charsets.
WorkstationID (WKID)	The workstation ID used by the client.
ApplicationName (APP)	The name used by System 10 to identify your application.
InterfacesFile (IFILE)	The pathname to the interfaces file.
OptimizePrepare (OP)	<p>OptimizePrepare={0 1 2}. This attribute determines whether stored procedures are created on the server for every call to SQLPrepare.</p> <p>When set to 0, stored procedures are created for every call to SQLPrepare. This setting can result in bad performance.</p> <p>When set to 1, the initial default, the driver creates stored procedures only if the statement contains parameters. Otherwise, the statement is cached and executed directly at SQLExecute time.</p> <p>When set to 2, the driver never creates stored procedures.</p>

## Using the Data Access Builder

Figure 170 (Page 2 of 3). Sybase System 10 Connection String Attributes

Attribute	Description
SelectMethod (SM)	<p>SelectMethod={0 1}. This attribute determines whether database cursors are used for Select statements. When set to 0, the initial default, database cursors are used. In some cases performance degradation can occur when performing large numbers of sequential Select statements because of the amount of overhead associated with creating database cursors.</p> <p>When set to 1, Select statements are executed directly without using database cursors. When set to 1, the data source is limited to one active statement and one active connection.</p>
ArraySize (AS)	<p>The number of rows the driver retrieves when fetching from the server. This is not the number of rows given to the user. This increases performance by reducing network traffic. The initial default is 10 rows.</p>
YieldProc (YLD)	<p>YieldProc={0 1 3}. This attribute determines whether you can work in other applications when Sybase System 10 is busy. This attribute is useful to users of ODBC applications.</p> <p>YieldProc=0 (peek and dispatch) causes the driver to check the Windows message queue and to send any messages to the appropriate Windows application.</p> <p>YieldProc=1 (no yielding, the initial default) does not let you work in other applications.</p> <p>YieldProc=3 (dispatch via Windows Yield function) turns control over to the Windows kernel. The Windows kernel checks the message queue and sends any messages to the appropriate application window.</p> <p>It is recommended that you use YieldProc=1. YieldProc=0 and YieldProc=3 do not work with all Windows applications.</p>
PasswordEncryption (PE)	<p>PasswordEncryption={0 1}. This attribute determines whether password encryption can be performed from the Open Client Library to the server (PasswordEncryption=1). When set to 0, the initial default, this cannot be done.</p>

## Using the Data Access Builder

Figure 170 (Page 3 of 3). Sybase System 10 Connection String Attributes

Attribute	Description
PacketSize (PS)	<p>PacketSize={-1 0 x}. This attribute determines the number of bytes per network packet transferred from the database server to the client. The correct setting of this attribute can improve performance.</p> <p>When set to 0, the initial default, the driver uses the default packet size as specified in the System 10 server configuration.</p> <p>When set to -1, the driver computes the maximum allowable packet size on the first connect to the data source and saves the value in the odbcc.ini file.</p> <p>When set to x, an integer from 1 to 10, which indicates a multiple of 512 bytes (for example, PacketSize=6 means to set the packet size to 6 * 512 = 3072 bytes).</p> <p>For you to take advantage of this connection attribute, you must configure the System 10 server for a maximum network packet size greater than or equal to the value you specified for PacketSize. For example,</p> <pre>sp_configure "maximum network packet size", 5120 reconfigure Restart System 10 Server</pre> <p>Note that the ODBC specification specifies a connect option, SQL_PACKET_SIZE, that offers this same functionality. To avoid conflicts with applications that may set both the connection string attribute and the ODBC connect option, they have been defined as mutually exclusive. If PacketSize is specified, you will receive a message "Driver Not Capable" if you attempt to call SQL_PACKET_SIZE. If you do not set PacketSize, then application calls to SQL_PACKET_SIZE are accepted by the driver.</p>

## Data Types

Figure 171 shows how the System 10 data types are mapped to the standard ODBC data types.

Figure 171 (Page 1 of 2). Sybase System 10 Data Types

System 10	ODBC
binary	SQL_BINARY
bit	SQL_BIT
char	SQL_CHAR
datetime	SQL_TIMESTAMP
decimal	SQL_DECIMAL

## Using the Data Access Builder

Figure 171 (Page 2 of 2). Sybase System 10 Data Types

System 10	ODBC
float	SQL_FLOAT
image	SQL_LONGVARBINARY
int	SQL_INTEGER
money	SQL_DECIMAL
numeric	SQL_NUMERIC
real	SQL_REAL
smalldatetime	SQL_TIMESTAMP
smallint	SQL_SMALLINT
smallmoney	SQL_DECIMAL
sysname	SQL_VARCHAR
text	SQL_LONGVARCHAR
timestamp	SQL_VARBINARY
tinyint	SQL_TINYINT
varbinary	SQL_VARBINARY
varchar	SQL_VARCHAR

## Isolation and Lock Levels Supported

System 10 supports isolation levels 1 (read committed, the default) and 3 (serializable). It supports page-level locking. See “Locking and Isolation Levels” on page 584 for a discussion of these topics.

## ODBC Conformance Level

The Sybase System 10 driver supports the Core, Level 1, and the following Level 2 functions:

- SQLDataSources
- SQLExtendedFetch(forward scrolling only)
- SQLMoreResults
- SQLNativeSql
- SQLNumParams
- SQLSetScrollOptions
- SQLBrowseConnect
- SQLColumnPrivileges
- SQLForeignKeys
- SQLPrimaryKeys
- SQLProcedureColumns

## Using the Data Access Builder

- SQLProcedures
- SQLTablePrivileges

The driver supports the minimum SQL grammar.

## Number of Connections and Statements Supported

The System 10 database system supports multiple connections and multiple statements per connection.

## **Using the Data Access Builder**



---


## Part 10. Defining National Characteristics

When you create applications for international markets, you can define nation- or locale-specific characteristics of the application separately, and then bind different sets of characteristics to your application for the different locales it will be used in.

This part of the *User's Guide* describes three utilities you can use in this process:

**ICONV, ICONVDEF** Convert a file from one code set to another

**LOCALDEF** Define a locale

For more information on binding resources to your application,  see Part 11, “Adding Application Resources” on page 613.

---

<b>Chapter 53. Code Set Conversion Utilities</b> . . . . .	605
ICONV Utility . . . . .	605
ICONVDEF Utility . . . . .	607
 <b>Chapter 54. LOCALDEF Utility</b> . . . . .	 609
LOCALDEF Options . . . . .	610
LOCALDEF Return Codes . . . . .	611

---





## Chapter 53. Code Set Conversion Utilities

This chapter describes the code set conversion utilities that help you convert a file from one code set to another.

- ICONV** Converts a file from one code set encoding to another. It can be used to convert C source code before compilation or to convert input files.
- ICONVDEF** Generates a translate table for use by the ICONV utility and `iconv` functions to perform code set conversion.

The ICONV utility calls the `iconv_open`, `iconv()`, and `iconv_close` functions to perform the code set translation at run time. These functions can be called from any program requiring code set translation. For more information on these functions, see the *C Library Reference*.

---

### ICONV Utility

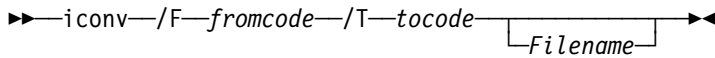
The ICONV utility converts the characters from the input file from one coded character set (code set) definition to another code set definition, and writes the characters to the output file (or **stdout** if no file is specified).

The ICONV utility uses the `iconv_open`, `iconv()`, and `iconv_close` functions to convert the input file strings from the coded character set definition for the input code page to the coded character set definition for the output code page. There is one string in the output file for each string in the input file. No padding or truncation of strings is performed.

When conversions are performed between single-byte code pages, the output strings are the same length as the input strings. When conversions are performed between double-byte code pages, the output strings may be longer or shorter than the input strings because the shift-out and shift-in characters may be added or removed.

## Code Set Conversion Utilities

The syntax of the `iconv` command is as follows:



where the options are:

**/F** Specify the input code set as *fromcode*

**/T** Specify the output code set as *tocode*

and you are converting code sets in *Filename*. If you do not specify *Filename*, ICONV reads from standard in (**stdin**). The converted file is sent to standard output (**stdout**).

If *Filename* contains codes that are not included in the *fromcode* code set, ICONV stops translating.

If *fromcode* specifies characters that do not exist in *tocode*, the characters are translated to a default character defined by the particular conversion taking place.

The values for *fromcode* and *tocode* identify either a converter within the converter DLL, or a conversion table created with ICONVDEF. See the *Introduction to Locale* chapter of the *Programming Guide* for details on supported conversions.

The function `iconv_open` is used to find a converter that matches the *fromcode* and *tocode* names.

ICONV opens the input and output files as binary. All characters from the input file (including any trailing `ctrl-Z` character) are converted from the input codepage to the output codepage and written to the output file.

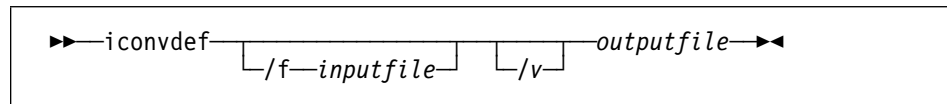
**Return Codes** The ICONV utility has the following return codes.

- 0** No errors were detected and the file was converted successfully.
- 1** An error occurred and the file was not converted.
- 2** An encoding error was encountered in the input file.

## ICONVDEF Utility

Use the ICONVDEF utility to create a conversion table that you can use with ICONV to convert a file from one code set to another.

The syntax of the `iconvdef` command is as follows:



ICONVDEF reads a source translation file from *inputfile* and writes the compiled version to *outputfile* (a conversion table). If you do not specify an input file, ICONVDEF uses standard input (**stdin**).

**/f** Indicates that the following term is *inputfile*.

**/v** Request more verbose information. Echoes input to **stdout**.

## Format of the Translation Source File

The translation source file can contain comment lines and directives. Start comment lines with the number sign (**#**). Directive lines have the following format:

*source target comment*

where:

*source* Specify the source bytes to be translated to *target*. Use hexadecimal values.

*target* Specify the value you want *source* translated to. Use either a hexadecimal value, or the keyword *invalid* to indicate that there is no valid target for the specified source.

*comment* Any additional text on the line is considered a comment, and ignored.

If you provide multiple assignments for the same *source*, only the last assignment is used.

Separate the *source*, *target*, and *comment* parameters with space or tab characters.

## ICONVDEF

### Return Codes

The ICONVDEF utility has the following return codes:


- 0** No errors were detected and the file was converted successfully.
- 1** An error occurred and the translation table was not successfully built.

## **Code Set Conversion Utilities**

## Chapter 54. LOCALDEF Utility

A *locale* is the definition of the subset of the user's environment that depends on language and cultural conventions. The locale object contains the rules and pointers to methods to implement the language and cultural conventions. A locale object is made up of a number of categories, identified by name, that control specific aspects of the behavior of components of the system.

The locale object is generated by the LOCALDEF utility according to the rules defined in the locale definition file. The locale object is implemented as a dynamic link library (DLL), but with the extension .LCL instead of .DLL. The locale can be loaded using the `setlocale()` function. Each .LCL file contains only one locale.

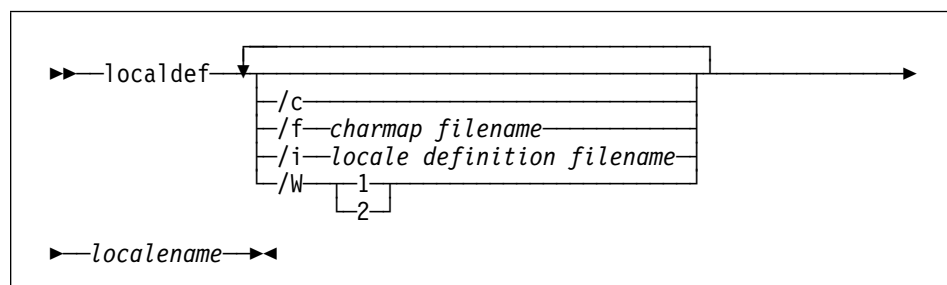
**Note:** Any locale object that you ship with your application will require CPPWMTHI.DLL, a VisualAge for C++ DLL. In order to ship this DLL with your application, you must rename it with the DLLRNAME utility.  See Chapter 47, “Packaging the VisualAge for C++ Runtime DLLs” on page 543 for more information.

The LOCALDEF utility reads the locale source and produces a locale object that can be used by the locale-specific library functions.

For additional information on internationalizing your applications, see the *Programming Guide*.

### Using LOCALDEF

Run the LOCALDEF utility with `localdef` command. The syntax of the `localdef` command is as follows:



where *localename* specifies the name of the output file for the locale generated. If you do not specify an extension for *localename*, LOCALDEF assumes the extension

## LOCALDEF Utility

.LCL. If you do not qualify *localename* with a path, the locale is written to the current directory.

You can use the following options:

- /C Generate locale even if there are errors
- /F Specify file that maps symbols to character encodings
- /I Specify locale source file
- /W Control messages produced

---

## LOCALDEF Options

### /C (Continue If Errors)

**Syntax:**

/C

**Default:**

Stop process when error encountered.

Use /C to generate a locale object even if LOCALDEF encounters an error during the locale definition process.

By default, LOCALDEF does not generate a locale object if there are errors.

### /F (Character Map)

**Syntax:**

/F *filename*

**Default:**

/F IBM-850.CM

Use /F to specify the name of the file that maps character symbols and collating element symbols to actual character encodings. If you do not specify an extension for the file name, LOCALDEF assumes .CM.

By default, LOCALDEF uses IBM-850.CM.



## LOCALDEF Utility

If you specify the file without including its path, LOCALDEF searches for it in the following places:

1. The current directory
2. The directories listed in the DPATH environment variable

### /I (Locale Source File)

**Syntax:**

/I

**Default:**

Standard input (**stdin**)

Use /I to specify the source file for the locale. If you do not specify an extension, LOCALDEF assumes the extension `.LOC`.

If you specify the file without including its path, LOCALDEF searches for it in the following places:

1. The current directory
2. The directories listed in the DPATH environment variable

### /W (Control Warnings)

**Syntax:**

/W[1|2]

**Default:**

/W2

Use /W to control the type of message LOCALDEF produces.

/W1 Produce severe errors and errors

/W2 Produce severe errors, errors, and warnings

By default, LOCALDEF produces all three kinds of messages.

---

## LOCALDEF Return Codes

The LOCALDEF utility has the following return codes.

- 0** No errors were detected and the locale was generated successfully.
- 1** Warning messages were issued and the locale was generated successfully.
- 2** The locale specification exceeded implementation limits or the coded character set or sets used were not supported by the implementation. The locale was **not** generated.
- >3** Warnings or errors were detected and the locale was **not** generated.

## **LOCALDEF Utility**

---

## Part 11. Adding Application Resources

This section of the *User's Guide* describes the Resource Workshop. Using the Resource Workshop, you can:

- Create and edit the resources of an application without using code
  - Compile your resources
  - Change a program's resources even when you don't have access to the source code
  - Check for errors, such as incorrect syntax and duplicate resource IDs
- 

<b>Chapter 55. Using Resource Tools</b> . . . . .	615
IRC.EXE: The Resource Compiler . . . . .	615
ILINK.EXE: The Resource Linker . . . . .	616
 <b>Chapter 56. An Overview of Working with Resources</b> . . . . .	617
What Is a Resource? . . . . .	617
Why You Should Link Resources to Your Applications . . . . .	619
Starting Resource Workshop . . . . .	619
Working with Resource Script Files . . . . .	620
Working with Graphic Files . . . . .	627
Customizing Resource Workshop . . . . .	628
 <b>Chapter 57. Working with Dialog Boxes</b> . . . . .	631
Editing a Dialog Box Resource Script . . . . .	631
Testing a Dialog Box Resource . . . . .	653
Customizing the Dialog Editor . . . . .	654
Using a Dialog Box Resource in an Application . . . . .	655
 <b>Chapter 58. Working with Menus and Accelerator Tables</b> . . . . .	657
Editing a Menu Resource Script . . . . .	657
Creating a Resource Script for a Floating Menu . . . . .	664
Customizing the Menu Editor . . . . .	664
Editing a Menuex Resource Script . . . . .	665
Editing an Accelerator Table Resource Script . . . . .	665
Customizing the Accelerator Editor . . . . .	668
Using Menu and Accelerator Resources in an Applciation . . . . .	668
 <b>Chapter 59. Working with String Tables</b> . . . . .	669
Editing a String Table Resource Script . . . . .	669
Customizing the String Editor . . . . .	671
Using a String Resource in an Application . . . . .	671

<b>Chapter 60. Working with Graphics</b> . . . . .	673
Editing a Resource Script for a Graphic . . . . .	673
Customizing the Graphic editor . . . . .	686
Using a Graphic Resource in an Application . . . . .	688
 <b>Chapter 61. Version Information and Custom Data Types</b> . . . . .	691
Creating a Custom Data Type . . . . .	691
Editing a Resource Script for Version Information or a Custom Data Type . . .	691
Editing a Resource Script for Version Information . . . . .	692
Compiling a Resource Script . . . . .	700
Using a Version Information Resource in an Application . . . . .	700
 <b>Chapter 62. Help Table and Help Subtable Resources</b> . . . . .	703
HELPTABLE Statement . . . . .	703
HELPSUBTABLE Statement . . . . .	704
Creating a Help Table Resource . . . . .	705
Creating a Help Subtable Resource . . . . .	705
Editing a Help Table or Help Subtable Resource . . . . .	706

---



## Chapter 55. Using Resource Tools

This chapter describes the following resource tools:

- `irc.exe` is the resource compiler. It is used to compile resource script files (.rc) files and produce a binary .res file.
- `ilink.exe` is the VisualAge for C++ linker which is used to bind resources, in .res file form, to an .exe file and marks the resulting .exe file as a Windows executable.

Windows programs provide a familiar and standard user interface for all applications. The components of the user interface are known as resources, and can include:

- Menus
- Dialog boxes
- Cursors
- Icons
- Version information
- Bitmaps
- String tables
- Accelerator keys
- Fonts
- User-defined data

Resources are defined external to your source code, and then attached to the executable file during linking. The application calls resources into memory only when needed, thus minimizing memory usage.

Resource script files (.rc) are text files that describe the resources a particular application uses. The resource compiler (`irc.exe`) uses the .rc file to compile the resources into a binary format resource (.res) file. `ILINK` then attaches the .res file, which contains your resources, to your executable.

---

### IRC.EXE: The Resource Compiler

IRC is a command-line version of Resource Workshop's resource compiler. It accepts a resource script file (.rc) as input and produces a resource object file (.res) as output. IRC uses the following command line syntax:

```
irc [options] filename.rc
```

The following is a summary of IRC options. Note that options are not case-sensitive (i.e. `-v` is the same as `-V`).

Option	Description
<code>-dname[=string]</code>	Defines a preprocessor symbol
<code>-fofilename</code>	Renames the output .res file. (By default, IRC creates the output .res file with the same name as the input .rc file.)

## Using Resource Tools

<b>-ipath</b>	Adds one or more directories (separated by semicolons) to the include search path
<b>-v</b>	Prints progress messages (verbose)
<b>-x</b>	Deletes the current include path
<b>-? or -h</b>	Displays switch help

Like Resource Workshop's resource compiler, IRC predefines common resource-related Windows constants, such as `WS_VISIBLE` and `BS_PUSHBUTTON`. Also, two special compiler-related symbols are defined: `RC_INVOKED` and `WORKSHOP_INVOKED`. These symbols can be used in the source text in conjunction with conditional preprocessor statements to control compilation. For example, the following construct can greatly speed up compilation:

```
#ifndef WORKSHOP_INVOKED
#include "windows.h"
#endif
```



The following example adds two directories to the include path and produces a .res file with the same name as the input .rc file.

```
irc -idir1;dir2 filename.rc
```


This example produces an output .res file with a name different from the input .rc file name:

```
irc -fofilename.res filename.rc
```

---

## ILINK.EXE: The Resource Linker

ILINK combines a .res file with an .exe file to produce a new Windows executable. ILINK accepts as input one or more object files (.res) and a single Windows executable file. ILINK links the resources by fixing up string tables and message tables and then binding these linked resources to the executable.

For more information on linking your resources with your application,  see Part 3, “Linking Your Program” on page 177.



## Chapter 56. An Overview of Working with Resources

Resource Workshop provides the tools you need to create and edit the resources of an application without using code. The tools include editors for dialog boxes, menus, accelerators, graphics, and strings. Resource Workshop also has a built-in script (text) editor, which you can use to create and edit non-graphical resources, such as version information, custom data types, help-tables, and help-subtables.

In general, you use Resource Workshop to perform the following tasks:

- Create and edit resource script (.rc) files. A resource script file contains the ASCII data from which you can generate one or more binary resources. Then you can link the resources to an application, and refer to the resources in the application's code.
- Edit resources in binary files. Resources are linked to an executable link library (.dll), or other binary file. When you open a binary file in Resource Workshop, the resources linked to the file are automatically decompiled so that you can edit them. When you are done making changes, you can recompile and relink the resources to the original binary file.  
**Note:** Resource Workshop is not able to decompile binary files containing menuex resources. If you decompile a binary file containing a menuex resource, incorrect results may be generated.
- Create and edit graphic files. Resource Workshop's Graphic editor is similar to the Windows Paintbrush. However, with the Graphic editor, you are not restricted to bitmap (.bmp) files; you can create icon (.ico), cursor (.cur), and font (.fnt) files as well.

This chapter describes resources and shows how to create and edit resources by working with resource script, and graphic files. This chapter also shows how to customize Resource Workshop.

---

### What Is a Resource?

A *resource* is a binary data item that is linked to an .exe, .dll, or other binary file. Typically, a resource defines one of the following user interface components:

- A *dialog box*, which is a pop-up window that uses labels, text boxes, buttons, check boxes, scroll bars, and other controls to give information to and receive information from a user. You can create a dialog box that pops up when a menu command is chosen or when an event occurs in an application. You can also create a dialog box that serves as the main window for an application.

## Overview of Working with Resources

- A *menu*, which is a list of commands from which a user can choose. Types of menus include standard menus, which appear on an application's menu bar; float menus, which can appear anywhere on the screen; and hierarchical (cascading) menus, which can appear when you choose a menu command. Applications often have standard menus that are labeled File, Edit, and Help; these menus contain file manipulation, editing, and online help commands, respectively.
- An *accelerator table*, which is one or more key combinations that a user can press to perform an action. For example, many applications use *Ctrl+X* to cut data and *Ctrl+V* to paste data.
- A *string table*, which is one or more text strings that can contain descriptions, prompts, error messages, and so on.
- A *bitmap*, which is a graphic image, such as a picture, logo, or other drawing.
- An *icon*, which is a graphic image that represents a minimized window. An icon is 64x64, 32x32, 16x32, or 16x16 pixels in size.
- A *cursor*, which is a graphic image that shows the position of the mouse on the screen and indicates the types of actions that a user can perform. A cursor is 32x32 pixels in size. Many applications use a pointer, which indicates that the user can make a selection, and an hourglass, which indicates that the system is performing an action.
- A *font*, which is a set of characters with a specific typeface and size. For example, **10-point Times Roman bold** is a font. A font resource can also contain a set of bitmaps that are not characters. Although Windows supports both raster fonts, which contain bitmaps for each character, and outline fonts, which contain drawing commands for each character, you can only create raster fonts in Resource Workshop. For more sophisticated font technology, you should consult with a third-party vendor.

A resource can also define:

- *Version information*, which is a block of data that can be used in several Windows API functions, including *GetFileVersionInfo* and *VerInstallFile*. Among other information, the version information resource specifies the version number, file type, and operating system of an application.
- A *custom data type*, which is any data type that you want to link to an application. For example, you can create a custom data type for blocks of text that exceed the 255-character limit of a string resource, or you can create a custom data type for metafiles, which are files that contain mathematical representations of images.



---

### Why You Should Link Resources to Your Applications

By linking resources to an application, you can keep data separate from code. This separation allows you to easily:

- Find and change parts of the application, such as user interface components. For example, if you keep all the strings for the application in a string table, you can easily customize the application for a foreign language; just open the resource script file containing the string table, translate the strings, save the resource script file, and then relink the corresponding resources to the application. You don't need to search for and change the strings in code.
- Change an application without changing its source files.
- Create and edit resources in a graphical environment. With Resource Workshop, you can work with resources just as they appear in the application. You don't need to compile and run the application to see user interface changes.

---

### Starting Resource Workshop

You create and edit resources using Resource Workshop. There are several different ways that Resource Workshop can be started.

To start Resource Workshop from the desktop:

- Open the **IBM VisualAge for C++ for Windows** group.  
Start Resource Workshop by double-clicking on the **Resource Workshop** icon.

To start Resource Workshop from WorkFrame:

- Double-click on a .rc, .cur, .ico, .fnt, .bmp, or .res resource object, or
- Click on a resource object using the right mouse button and select **Edit resource** from the popup menu, or
- Select a resource object by clicking on it once with the left mouse button and choose **Selected→Edit resource**.


After you have started Resource Workshop, you are ready to create and edit resources by working with resource script and graphic files.


**To ...**

Create or edit a resource script file

Create or edit a graphic file

**See ...**

 “Creating a Resource Script File” on page 620

 “Working with Graphic Files” on page 627

## Overview of Working with Resources


---

### Working with Resource Script Files

To create and edit resources, you use resource script files. A *resource script file* contains one or more sets of ASCII data, called *resource scripts*, from which binary resources can be generated. Each resource type has a different resource script format; however, you don't need to know these formats unless you plan to edit a resource script file in a text editor. For information about resource script formats, search online help for "Editing a resource as text."

In Resource Workshop, you can create and edit resource scripts using the Dialog, Menu, Accelerator, Graphic, String, and Script editors. Use the Graphic editor to work with bitmaps, cursors, icons, and fonts; use the Script editor to work with raw resource scripts, including resource scripts for version information and custom data types.

Unlike a text editor, most Resource Workshop editors are graphical; they allow you to work with resources just as they appear in an application. When you save your work in Resource Workshop, you can create a resource script file automatically. The file contains resource scripts that correspond to the resources you define in the Resource Workshop editors.

In addition to resource scripts, a resource script file can contain one or more identifiers. An *identifier* is a unique name or number that you can use in code to refer to a resource. In Resource Workshop, you can have identifiers created automatically when you add new resources, or you can create identifiers manually. For more information about automatically creating identifiers,  see "Setting General Environment Options" on page 629.

**Note:** You can't refer to a resource in code unless the resource has a corresponding identifier.

The following sections show how to

- Create a new resource script file
- Edit an existing resource script file
- Generate binary resources from a resource script file

### Creating a Resource Script File


To create a resource script file:

1. Choose **File→New resource project**.
2. Select **.RC**.
3. Choose **OK**.

The **Add file to resource project** dialog box appears.

## Overview of Working with Resources

4. If you want to store identifiers in a separate file, type the name of a resource header file in the **File name** box, and choose **OK**. (Just like standard C header files, the resource header file can contain identifiers.) Otherwise, choose **Cancel**.

You can add resource header files later by using the **File→Add to resource project** command. For more information,  see “Adding an Identifier” on page 625.

5. Choose **File→Save resource project**.
6. In the **New file name** box, type a name for the new resource script file.
7. Choose **OK**.

Now you have an empty resource script file to which you can add resource scripts and identifiers.

### Editing an Existing Resource Script File

To edit an existing resource script file:

1. Choose **File→Open resource project**.
2. In the **File type** box, choose **RC resource script**.
3. In the **File name** box, type the resource script file name, or select the file name using the **Files** and **Directories** boxes.
4. Choose **OK**.
5. Add, edit, or remove resource scripts and identifiers from the resource script file. These procedures are described in the following sections.
6. Choose **File→Save resource project**.

### Adding Resource Scripts

You can add resource scripts to a resource script file directly or indirectly. When you add resource scripts to a resource script file *directly*, the corresponding data is included in the current resource script file. When you add resource scripts *indirectly*, you can access the corresponding data from the current resource script file, but the data is stored in a separate resource script file.

If you want to keep your resource scripts together, you should add them directly to a single resource script file. On the other hand, if you want to share resource scripts among several resource script files, you should add the resource scripts indirectly to each file.

**Adding a New Resource Script Indirectly** To add a new resource script indirectly:

1. Create or open the resource script file.
2. Choose **File→Add to resource project**.

## Overview of Working with Resources

3. In the **File type** box, choose **RC resource script**.
4. Type a name for the new resource script file. The file name must be unique from other files in the current directory.
5. Choose **OK**.

A dialog box asks whether you want to create a new resource script file.
6. Choose **Yes**.

### Adding One or More Existing Resource Scripts Indirectly


To add one or more existing resource scripts indirectly:

1. Open the resource script file.
2. Choose **File→Add to resource project**.
3. In the **File type** box, type the name of the file that contains the resource script(s). Or select the file using the **Files** and **Directories** boxes. You can specify a resource script, binary, or graphic file. All resource scripts in the file will be added indirectly to the current resource script file.
4. Choose **OK**.

**Note:** If a file contains more resource scripts than you need, you can use the **Resource→Save resource as** command to separate some resource scripts into new files. Then you can add the new files indirectly to the current resource script file.

### Adding a New Resource Script Directly

To add a new resource script directly:

1. Create or open the resource script file.
2. Choose **Resource→New**.
3. In the **Resource type** box, select a resource type.
4. Choose the resource script file to which you want to add the new resource script. You can choose the current resource script file or any file containing resource scripts that have been added indirectly to the current project.
5. Choose **OK**.
6. If you are embedding a definition for a graphic, choose **Source**. You can also add the graphic to the current resource script file by choosing **Binary**, but this choice adds the new graphic *indirectly*.
7. If you are embedding a definition for a bitmap or icon, specify size and color options, and then choose **OK**. If you don't know which size and color options to specify, accept the default values. For information about changing the size and color options,  see “Changing the Properties of a Bitmap” on page 682 or “Changing the Properties of an Icon” on page 683.

## Overview of Working with Resources

### Adding an Existing Resource Script Directly

To add an existing resource script directly, copy or move the resource script from one file to another.

To copy or move a resource script:

1. Open the file that contains the resource script. (You might want to save the current file first.)
2. In the **Resource Project** window, select the resource script.
3. Choose **Edit→Copy** (to copy the resource script) or **Edit→Cut** (to move the resource script).
4. Open the file to which you want to copy or move the resource script.
5. Choose **Edit→Paste**.

**Note:** You can only paste a resource script into a file with a compatible format. For example, you can't paste an icon script into a bitmap file.

### Editing a Resource Script

You can edit a resource script by using one of the Resource Workshop editors or by specifying memory or language options.


To edit a resource script in a Resource Workshop editor:

1. In the **Resource Project** window, double-click the resource script.
2. Follow the instructions in the appropriate chapter of this book (see the following table).


#### For information about working with ...

#### See...


Dialog boxes

 Chapter 57, “Working with Dialog Boxes” on page 631


Menus and accelerators

 Chapter 58, “Working with Menus and Accelerator Tables” on page 657


Strings

 Chapter 59, “Working with String Tables” on page 669

Bitmaps, icons, cursors, and fonts

 Chapter 60, “Working with Graphics” on page 673

Version information and custom data types

 Chapter 61, “Version Information and Custom Data Types” on page 691

## Overview of Working with Resources

### Specifying Memory Options

Each resource script includes information about how the resource is handled in memory.

To specify memory options for a resource script:

1. In the **Resource Project** window, select the resource script.
2. Choose **Resource→Memory options**.
3. Select or deselect the following memory options:

Option	Description
Loaded on call	The resource is loaded into memory when needed for the first time. If a resource is not loaded on call, it is loaded when the application starts. If an application doesn't need the resource immediately, select this option so that the application is launched more quickly.
Moveable	The resource can be moved in memory. By allowing a resource to be moved in memory, you can increase the efficiency with which the application uses memory; however, it is harder to track the exact location of the resource.
Discardable	The resource can be removed from memory. (If no longer in memory when needed, the resource is automatically reloaded.) By allowing a resource to be discarded from memory, you can decrease the amount of memory an application needs at one time; however, if a system runs low on memory, the application will run more slowly when reloading resources.
Pure	The resource cannot be changed in memory.

4. Choose **OK**.

### Specifying a Language Version

For a Win32 application, you can create resources with different versions for each language, such as French, German, Spanish, and so on. A Win32 application automatically uses the resources that correspond to the system on which it runs. For example, a Win32 application running on a French system automatically uses the French versions of resources. (To set the system language, use the **International settings** in the Windows Control Panel.)

To specify a language version for a Win32 resource script:

1. In the **Resource Project** window, select the resource script.
2. Choose **Resource→Language**.

## Overview of Working with Resources

3. Choose a major language or **Neutral**. (A resource with a Neutral language version works on all systems.)

A major language is a language with one or more dialects. For example, the English major language has dialects for Australia, Canada, New Zealand, the United Kingdom, and the United States.

4. Choose a minor language or **Neutral**.

A minor language is a particular dialect within a major language.

5. Choose **OK**.

### Removing a Resource Script

To delete a resource script from a resource script file:

1. In the **Resource Project** window, select the resource script.
2. Press Delete.

### Adding an Identifier

If you are not generating identifiers automatically or if you want to create identifiers for resources that do not exist yet, you need to add identifiers manually.

To add an identifier to a resource header or resource script file:

1. Choose **Resource→Identifiers**.
2. Choose **New**.
3. In the **Name** box, type a name for the identifier. You can use alphanumeric characters and the underscore character. The identifier name cannot begin with a number, and any characters past the 31st character are ignored.
4. In the **Value** box, type a value for the identifier. You can choose any value that is not already used.
5. Choose a resource script or header file in which to store the identifier.
6. Choose **OK**.
7. Choose **Done**.

You can also add an existing identifier by moving it from one resource script or resource header file to another. To move an identifier, choose **Resource→Identifiers** and use the **Move** button.

**Note:** You can control which information appears in the **Identifiers** dialog box by setting the **View** and **Sort** options. For more information about these options, choose **Resource→Identifiers**, and then choose **Help**.

## Overview of Working with Resources

Depending on how you want to organize identifiers and which identifiers you have already defined, you might want to add a new or existing resource header file to the current resource script file.

To add a new or existing resource header file to the current resource script file:

1. Choose **File→Add to resource project**.
2. In the **File type** box, choose **RH, H c Header**.
3. In the **File name** box, type the name of the new or existing resource header file.
4. Choose **OK**.
5. If you are adding a new resource header file, a dialog box warns you that the resource header file doesn't currently exist. Choose **Yes** to create the new resource header file.

### Editing an Identifier

You can edit a resource identifier by changing its value or name. You may want to change the value of an identifier to keep related resources together; you might want to change the name of an identifier to more accurately reflect the resource to which it refers.

To edit a resource identifier:

1. Choose **Resource→Identifiers**.
2. Select the identifier.
3. Choose **Change** or **Rename**.
4. Type a new value or name for the resource identifier.
5. Choose **OK**.
6. Choose **Done**.

**Note:** You can control which information appears in the **Identifiers** dialog box by setting the **View** and **Sort** options. For more information about these options, choose **Resource→Identifiers** and then choose **Help**.

### Removing an identifier

If you remove a resource script from a resource script file, you should also remove its identifier.

To remove a resource identifier,

1. Choose **Resource→Identifiers**.
2. Select the identifier.
3. Choose **Delete**.



## Overview of Working with Resources

4. If the identifier is assigned to a resource, a warning box appears. Choose **Yes** to delete the identifier.
5. Choose **Done**.

**Note:** You can control which information appears in the **Identifiers** dialog box by setting the **View** and **Sort** options. For more information about these options, choose **Resource→Identifiers**, and then choose **Help**.

### Generating Binary Resources from a Resource Script File


After you add one or more resource scripts and identifiers to a resource script file, you can compile the resource script file into one or more binary resources, link the binary resources to an application, and then refer to the resources in the application's code by using the corresponding identifiers.

You can generate resources on the command line with the resource compiler (`irc.exe`) and the resource linker (`ilink.exe`).

**Note:** To refer to resources in code, you must include the corresponding resource header file(s) in your C or C++ source files.

---


## Working with Graphic Files

A graphic file is a bitmap (.bmp), icon (.ico), cursor (.cur), or font (.fnt) file. You can add a graphic file to a resource script file to create a graphic resource, which you can then use in an application. For more information about adding a graphic file to a resource script file,  see “Adding Resource Scripts” on page 621.

You can also use graphic files in many applications besides Resource Workshop. For example, you can include bitmaps in most word-processor documents.

### Creating a Graphic File

To create a graphic file:

1. Choose **File→New resource project**.
2. Select a graphic file type.
3. Choose **OK**.
4. If you are creating a bitmap or icon file, specify size and color options, and then choose **OK**. If you don't know which size and color options to specify, accept the default values. For information about changing the size and color options,  see “Changing the Properties of a Bitmap” on page 682 or “Changing the Properties of an Icon” on page 683.
5. Choose **File→Save resource project**.


## Overview of Working with Resources

6. In the **New file name** box, type a name for the new graphic file.
7. Choose **OK**.

You can also create a graphic file by opening a binary or resource script file that contains a graphic, selecting the graphic in the **Resource Project** window, and then saving the graphic in a graphic file using the **Resource→Save resource as** command.

## Editing a Graphic File

To edit a graphic file:

1. Choose **File→Open resource project**.
2. In the **File type** box, choose the graphic file type.
3. In the **File name** box, type the graphic filename. Or select the filename using the **Files** and **Directories** boxes.
4. Choose **OK**.
5. Use the graphic editing tools, which are described in  Chapter 60, “Working with Graphics” on page 673 .
6. Choose **File→Save resource project**.

---

## Customizing Resource Workshop

You can customize Resource Workshop by specifying which information appears in the **Resource Project** window, and by setting general environment options.

### Specifying Which Information Appears in the Resource Project Window

If there is an open file in Resource Workshop, you can specify which information appears in the **Resource Project** window by using the following commands:

Command	Description
<b>View→By type</b> or <b>View→By file</b>	Group resource scripts by type (dialog boxes, menus, accelerators, and so on), or by file (first file, second file, and so on).
<b>View→Show identifiers</b>	Show the identifiers in the current resource script file.
<b>View→Show resources</b>	Show the resource scripts in the current resource script file.
<b>View→Show items</b>	Show individual items in resource scripts, such as the individual commands on menus, strings in a string table, and accelerators in an accelerator table.

## Overview of Working with Resources

**View→Show unused types** Show placeholders in the **Resource Project** window for resource types that have not been added to the current resource script file.

**View→Show vertical preview**, **View→Show horizontal preview**, or **View→Hide preview**  
Hide the **Preview** pane of the **Resource Project** window or display the **Preview** pane horizontally or vertically. (The **Preview** pane shows a miniature version of the selected resource as it appears in the corresponding Resource Workshop editor.)

### Setting General Environment Options


To set general environment options:

1. Choose **File→Preferences**.
2. Set any of the following options:

Option	Settings
<b>Undo levels</b>	1-99. The maximum number of Undo levels may be limited by available memory.
<b>Text editor</b>	The full path and file name of a text editor. You will be given the option of editing the resource script file in the text editor if an error occurs during the resource compilation process. By default, the text editor used is the VisualAge for C++ Editor.
<b>Include path</b>	The path(s) where Resource Workshop looks for resource files. You can specify the include path only when there are no open files.
<b>Multi-save</b>	Whether or not the current resource file is saved automatically in an additional compiled resource script (.res), or backup file. (Backup files have a tilde (~) as the first character in the file name extension.)
<b>Automatic identifier management</b>	Whether or not identifiers are automatically created when you add new resource scripts. If you turn on this feature, you will not have to manually create identifiers for new resource scripts.

## Overview of Working with Resources

### Language for Win32

The default major and minor languages for new resource scripts. You can specify the default languages only after targeting Win32. For more information,  see “Specifying a Language Version” on page 624.

3. Choose **OK**.

## Chapter 57. Working with Dialog Boxes

A *dialog box* is a pop-up window that uses labels, text boxes, buttons, check boxes, scroll bars, and other controls to give information to and receive information from a user. You can create a dialog box that pops up when a menu command is chosen or when an event occurs in an application. You can also create a dialog box that serves as the main window of an application.

Like other resources, you define a dialog box with a resource script. This chapter describes how to edit and test a dialog box resource script, and how to use the corresponding resource in an application.

---

### Editing a Dialog Box Resource Script

You edit a dialog box resource script by using the Dialog editor.

To start the Dialog editor,

1. Create a resource script file and add a dialog box resource script, or open a resource script file that contains a dialog box resource script.
2. If the Dialog editor doesn't start automatically, double-click a dialog box resource script in the **Resource Project** window.

The Dialog editor appears. It contains a dialog box template, **Tools** palette, and **Alignment** palette.

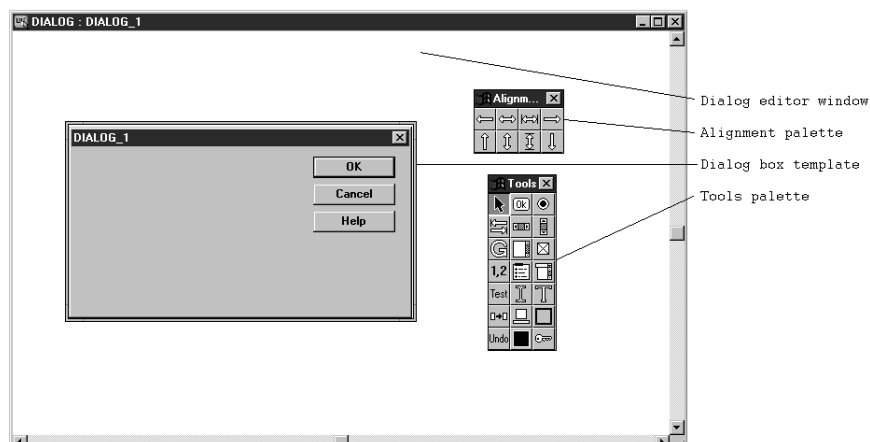


Figure 172. Dialog editor

## Working with Dialog Boxes

The **Tools** palette has the following tools:

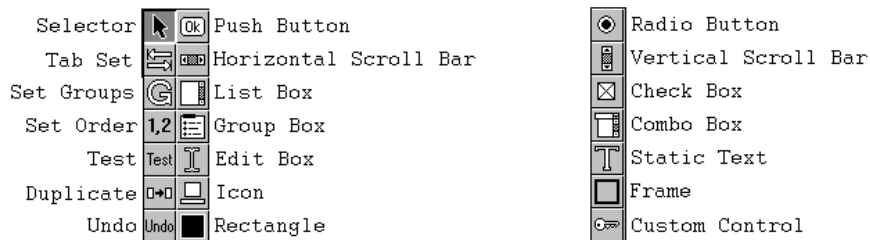


Figure 173. Dialog editor - Tools palette

The **Alignment** palette has the following tools:

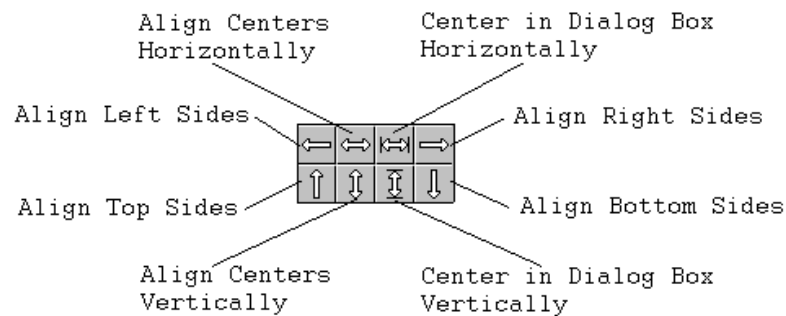



Figure 174. Dialog editor - Alignment palette

After you start the Dialog editor, you can edit a dialog box resource script by

- Setting dialog box properties
- Moving and resizing the dialog box
- Adding controls to the dialog box
- Moving, resizing, aligning, and grouping controls
- Setting the Windows classes of controls
- Setting control properties

To customize the Dialog editor,  see “Customizing the Dialog Editor” on page 654.

### Setting Dialog Box Properties

To set the properties of a dialog box, double-click its title bar or outer edge, and then use the **Window style** dialog box. You can set any of the properties described in the following sections.

## Working with Dialog Boxes


### Class Property

You can specify your own custom window class. If you don't specify a class, the dialog box uses the standard Windows class.

For information about creating Window classes, search Win32 Online Help (win32.hlp) for "Windows Classes."

### Menu Property

Like other windows, a dialog box can have menus. Menus are particularly useful for dialog boxes that serve as the main window of an application.

To add menus to your dialog box, specify the identifier or unique number of a menu resource. For information about menu resources,  see Chapter 58, "Working with Menus and Accelerator Tables" on page 657.

### Type Property

You can set the **Type** property to one of the following values:

Value	Description
<b>Popup</b>	A pop-up window. Because most dialog boxes are pop-ups, this is the default.
<b>Child</b>	A child of the current window
<b>Overlapped</b>	A window that can be covered by another window. You will want to set the <b>Type</b> property to <b>Overlapped</b> only for the main window in an application.

### Style Properties

You can turn the following **Style** properties on or off:

Property	Description
<b>System menu</b>	Includes a System menu box to the left of the title bar (appears only if you have also chosen <b>Caption</b> for the <b>Frame Style</b> property). If the dialog box is a child window, a <b>Close</b> box appears instead of a Control menu.
<b>Thick frame</b>	Places a thick frame around the dialog box. This option defines what the user will see when the dialog box appears within an application. If you want the dialog box to be resizable, use this option. (Don't confuse this option with the <b>Thick frame</b> option in the Dialog editor <b>Preferences</b> command. That option defines what the dialog box looks like when you select it in the Dialog editor.)
<b>Vertical scroll</b>	Adds a vertical scroll bar to the dialog box frame
<b>Horizontal scroll</b>	Adds a horizontal scroll bar to the dialog box frame

## Working with Dialog Boxes

<b>Minimize box</b>	Adds a <b>Minimize</b> button on the right side of the title bar. The <b>Minimize</b> button appears only if you have also chosen <b>Caption</b> for the dialog box <b>Frame Style</b> .
<b>Maximize box</b>	Adds a <b>Maximize</b> button on the right side of the title bar. The <b>Maximize</b> button appears only if you have also chosen <b>Caption</b> for the dialog box <b>Frame Style</b> .
<b>Absolute align</b>	Makes the dialog box coordinates relative to the display screen rather than the parent or owner window
<b>System modal</b>	Makes the dialog box system modal, meaning that the user can't do anything else until the dialog box is closed
<b>Local edit</b>	Allocates any edit text controls included in this dialog box to the application's local heap. Choose <b>Local edit</b> if your application needs to use EM_SETHANDLE and EM_GETHANDLE messages.
<b>Modal frame</b>	Frames the dialog box with a combination of the dialog box <b>Frame</b> and <b>Caption Styles</b> (default)
<b>No idle messages</b>	Suppresses sending WM_ENTERIDLE messages to the application's main dialog box. The dialog box must be modal for this option to take effect.
<b>Clip children</b>	Protects the client area of child dialog boxes from being drawn on by the parent dialog box
<b>Clip siblings</b>	Protects the siblings of this window. Drawing is restricted to this window. This option is not required for dialog boxes, but can be useful for child dialog windows.
<b>Visible</b>	Makes a modeless window visible before the return from CreateDialog. This option has no effect on modal windows (the usual kind of dialog box). By default, this option is not checked (NOT WS_VISIBLE).

**Frame Style** You can set the **Frame style** property to one of the following values:

<b>Property</b>	<b>Value</b>	<b>Description</b>
	<b>Dialog frame</b>	A double border without a title bar
	<b>Border</b>	A single, thin border without a title bar
	<b>Caption</b>	A single, thin border and a title bar where a caption can be displayed (default)
	<b>No border</b>	No border or title bar



## Working with Dialog Boxes

### Font Properties


You can set the **Font Name** and **Font Size** properties by choosing the **Fonts** button, selecting a font name and size, and then choosing **OK**.

### Extended Style Property


You can set the **Extended Style** property to one of the following values:

Value	Description
<b>WS_ES_ACCEPTFILES</b>	This dialog box accepts drag and drop files
<b>WS_EX_DLGMODALFRAME</b>	This dialog box has a double border that can (optionally) be created with a title bar by specifying the <b>WS_CAPTION</b> style flag in the c-style parameter
<b>WS_EX_NOPARENTNOTIFY</b>	A child window created by using this style will not send the <b>WM_PARENTNOTIFY</b> message to its parent window when the child window is created or destroyed
<b>WS_EX_TOPMOST</b>	A window created with this style should be placed above all non-topmost windows and stay above them even when the window is deactivated. An application can use the <i>SetWindowPos</i> function to add or remove this attribute.
<b>WS_EX_TRANSPARENT</b>	A window created with this style is transparent, so that any windows beneath it are not obscured by the window. A window created with this style receives <b>WM_PAINT</b> messages only after all sibling windows beneath it have been updated.

### Common Properties

In addition, you can set any of the common properties for the dialog box. For more information,  see “Setting Common Properties using the Properties Window” on page 644.

### Moving and Resizing a Dialog Box

You can move or resize a dialog box just as you would move or resize a control. For more information,  see “Moving Controls” on page 638 and “Resizing Controls” on page 638.

If you set the **Type** property of the dialog box to **Overlapped**, you can let Windows control the position of the dialog box on the screen. Select the dialog box, choose **Align→Size**, select the **Set by Windows** box, and then choose **OK**.

## Working with Dialog Boxes


### Adding a Standard Control to a Dialog Box

To add a standard control to a dialog box,

1. In the **Tools** palette, select the control. (The controls are described in a table in this section.)
2. Drag the mouse in the dialog box template.

If you select a control from the **Tools** palette and then change your mind about placing it, press Esc.

You can add any of the following controls:

Control	Description
<b>Push button</b>	A rectangular button the user "presses" to select an action
<b>Radio button</b>	A button that is used in groups to represent mutually exclusive options. A radio button is circular. For information about grouping radio buttons,  see the section "Grouping and Ungrouping Controls" on page 640.
<b>Check box</b>	A square box that represents an option that can be turned on or off. When a check box is selected, an X appears in the square. When a check box is not selected, the square is empty.
<b>Static text</b>	A fixed text string
<b>Group box</b>	A box that is used to group controls visually
<b>Scroll bar</b>	A rectangle with direction arrows at each end. Between the arrows, a square icon, called the scroll box or thumb, indicates the approximate position of the display relative to the full range of information. You can create a horizontal or vertical scroll bar.
<b>Edit box</b>	A box in which the user can edit text
<b>List box</b>	A box that contains a list of items from which the user can choose
<b>Combo box</b>	A box in which a user can type or select an item
<b>Icon</b>	An icon
<b>Frame</b>	A rectangular frame with the same color as the dialog box frame
<b>Rectangle</b>	A rectangle with the same color as the dialog box frame
<b>Line</b>	A line used to separate items in a dialog box. You can create a horizontal or vertical line.

## Working with Dialog Boxes

### Working with Custom Controls

In addition to standard controls, you can add custom controls to a dialog box template. A custom control is any other window class that you want to add to your dialog box. You can create your own custom controls or buy custom controls from a third-party vendor.

#### Adding a Custom Control to a Dialog Box

To add a custom control to a dialog box:

1. Click the **Custom Control** tool or choose **Control→Custom**.
2. In the **Class** box, choose the custom control.

Resource Workshop displays a sample of the custom control you have selected in the middle of the dialog box.

3. Choose **OK**.
4. In the dialog box template, drag the mouse where you want to place the custom control.

An icon representing the custom control appears in the dialog box template.

#### Creating Custom Controls Selecting Controls

To create custom controls, you must design and store them in dynamic-link library (.DLL) files.

Many editing options require that one or more controls be *selected*. For example, you must select a control before you can change its size, and you must select at least two controls before you can align them relative to each other.

To select a single control,

1. Choose the **Selector** tool.
2. Click the control.

A selection frame appears around the control you select.


To select one or more controls,

1. Choose the **Selector** tool.
2. Click on a control to make sure the dialog box template is not selected.
3. Drag a rectangle around the controls you want to select. Depending on the current state of the **Selection rectangle surrounds** check box in the **Options→Preferences** dialog box, the selection rectangle must either entirely surround the controls or just touch them.

A selection frame appears around the controls you select.

## Working with Dialog Boxes

You can also perform the following selection actions:

To	Perform this action
Add or remove a single control from a selected group	Shift+click the control.
Select all the controls in a dialog box	Choose <b>Edit</b> → <b>Select all</b> .
Move the selection frame	Press Tab (to select the next control in the tab order) or Shift+Tab (to select the previous control in the tab order). If the current selection includes more than one control, the selection frame moves to the control that precedes or follows the highest one in the current selection. However, if the current selection includes controls selected with Shift+click, the selection frame moves to the control that precedes or follows the last control you selected. For more information about the tab order,  see “Setting the Tab Order of Controls” on page 640.

## Moving Controls

You can move selected controls by dragging anywhere inside the selection frame. As you drag, the controls move together, maintaining their positions relative to each other.

To move a control with the keyboard, press Tab to select the control, then use the arrow keys to move the control, and press Enter.

## Resizing Controls

You can resize controls with the options in the **Size Controls** dialog box:

1. Select the controls you want to resize.

The selection frame surrounds the selected controls. The individual controls that are affected by the resizing options have shaded outlines.

2. Choose **Align**→**Size**. Choose the **Vertical size** and **Horizontal size** options you want and click **OK** to resize the selected controls.

Horizontal size options	Description
<b>No change</b>	Prevents horizontal size change
<b>Shrink to smallest</b>	Reduces width of controls to match the least wide of the selected controls

## Working with Dialog Boxes

<b>Grow to largest</b>	Increases width of controls to match the widest of the selected controls
<b>Width of size box</b>	Resizes controls so they are as wide as the selection frame
<b>Width of dialog</b>	Resizes controls so they are as wide as the dialog box

Vertical size options	Description
<b>No change</b>	Prevents vertical size change
<b>Shrink to smallest</b>	Reduces height of controls to match the least tall of the selected controls
<b>Grow to largest</b>	Increases height of controls to match the tallest of the selected controls
<b>Height of size box</b>	Resizes controls so they are as tall as the selection frame
<b>Height of dialog</b>	Resizes controls so they are as tall as the dialog box

If you select a single control and then choose **Align→Size**, the following options are available:

- **No change** (Horizontal and Vertical)
- **Width of dialog** (Horizontal)
- **Height of dialog** (Vertical)
- **Enter values** (Horizontal and Vertical)

The **No change**, **Width of dialog**, and **Height of dialog** options are described in the preceding tables.

The **Enter values** options let you specify both the size and position of the selected control. The X- and Y-values set the distance of the upper left corner of the control from the upper left corner of the dialog box (directly below the title bar). The CX- and CY-values set the width and height of the control. All values are measured in pixels or dialog units.

### Fine-Tuning the Size of Controls

You can use the keyboard and mouse together to fine-tune the size of a control:

1. Select the control and move the mouse pointer over an edge or corner of the control.

## Working with Dialog Boxes

The cursor becomes a double-headed arrow.

2. Hold down mouse button one and press an arrow key. Each time you press the arrow key, the selection frame moves a single pixel or dialog unit.

## Moving and Resizing Controls at the Same Time

You can specify the position and size of a control at the same time:

1. Select the control.
2. Choose **Align→Size** or hold down the Alt key while double-clicking the mouse.

The **Size Controls** dialog box appears.

3. To set the position of the upper left corner of the control, specify its X- and Y-coordinates in pixels or dialog units. The coordinates 0,0 place the control in the upper left corner of the dialog box, directly below the title bar.
4. To set the control's width and height, specify its CX- and CY-values in pixels or dialog units.

## Grouping and Ungrouping Controls


You can group controls to let the user move among them using the arrow keys.

To group controls,

1. Click the **Set Groups** tool or choose **Options→Set groups**.
2. Click a control that is not surrounded by a shaded outline.

To ungroup controls,

1. Click the **Set Groups** tool or choose **Options→Set groups**.
2. Click a control that is surrounded by a shaded outline.

You can also set the **Group** property using the **Style** window. For more information about the **Style** window,  see “Setting Common Properties using the Style Window” on page 645.

**Note:** You mark only the *first* control in each group with the **Group** attribute. Controls are included in the group according to the order in which they are added to the dialog box. For example, if you add seven controls to your dialog box and then turn on the **Group** property of the first and fifth controls, the first four controls are included in the first group, and the remaining controls are included in the second group.

## Setting the Tab Order of Controls

You can specify the order in which users can tab to the controls in a dialog box.

To specify the tab order of the controls in your dialog box,

## Working with Dialog Boxes

1. Select the controls whose tab order you want to change. To specify the order for *all* controls in the dialog box, don't select any controls.

2. Click the **Set Order** tool. Or choose **Options→Set order**.

Each control you selected is numbered to show its current place in the overall order. Note the **Next item** prompt at the bottom of the Dialog editor. It tells you the number that will be assigned to the next control you select.

3. Click the controls to which you want to assign numbers.

While assigning new order numbers, you can "step back" by relicking the last control you just clicked. The order changes to its previous number. You can continue to backtrack by clicking the controls in the reverse order that you originally clicked them.

4. When you are done assigning numbers, click the **Selector** tool to return to selection mode.

### Aligning Controls

You can align controls on the dialog box template so they are more visually appealing. You align controls with the **Align→Align** command, the **Alignment** palette, and the grid. These alignment methods are described in the following sections.

#### Aligning controls with the **Align→Align** command

To align your controls with the **Align→Align** command,

1. Select the controls you want to align.

A selection frame surrounds all the selected controls. The individual controls you can align are shaded.

2. Choose **Align→Align**. The **Align Controls** dialog box appears.
3. Choose the **Vertical alignment** and **Horizontal alignment** options you want, and then choose **OK**.

Horizontal alignment options	Description
<b>No change</b>	Affects no change in horizontal alignment
<b>Left sides</b>	Aligns the controls so their left sides are on the left of the selection frame
<b>Centers</b>	Aligns the controls so their horizontal centers are in the center of the selection frame
<b>Right sides</b>	Aligns the controls so their right sides are on the right side of the selection frame

## Working with Dialog Boxes

<b>Space equally</b>	Moves the controls horizontally within the selection frame so the spaces between them are equal
<b>Center in dialog</b>	Centers the selection frame horizontally in the dialog box. The relative position of the individual controls within the selection frame is unchanged.

<b>Vertical alignment options</b>	<b>Description</b>
<b>No change</b>	Affects no change in vertical alignment
<b>Tops</b>	Aligns the controls so their tops are at the top of the selection frame
<b>Centers</b>	Aligns the controls so their vertical centers are in the center of the selection frame
<b>Bottoms</b>	Aligns the controls so their bottoms are at the bottom of the selection frame
<b>Space equally</b>	Moves the controls vertically within the selection frame so the spaces between them are equal
<b>Center in dialog</b>	Centers the selection frame vertically in the dialog box. The relative position of the individual controls within the selection frame is unchanged.

### Aligning Controls with the Alignment Palette

You can also align controls using the **Alignment** palette. Select the controls you want to align and then choose a tool from the **Alignment** palette.

The **Space equally** option is not in the **Alignment** palette. However, you can use the **Align Controls** dialog box as explained in the previous section, or you can space controls equally by stretching the selection frame.

To space controls equally by stretching the selection frame,

1. Select the controls you want to space equally.  
A selection frame surrounds your selected controls.
2. Stretch the selection frame by holding down the Ctrl key while dragging the edge of the selection frame in the direction you want to space your controls:



## Working with Dialog Boxes

- To space the controls equally in the horizontal direction, stretch the right or left border of the selection frame.
- To space the controls equally in the vertical direction, stretch the top or bottom border of the selection frame.

If you stretch a *corner* of the selection frame, the **Form Controls into an Array** dialog box appears, ready for you to arrange your controls in rows and columns.

### Aligning Controls with the Grid

To align controls with the grid:

1. Choose **Align→Grid**. The **Set Grid Attributes** dialog box appears.
2. Specify the width and height of a grid cell (in dialog units).
3. Specify the **Grid type**:
  - **Absolute**. Snaps the control to the nearest grid line.
  - **Relative**. Moves the control only in increments of the grid width horizontally and the grid height vertically. Therefore, if a control was not placed on a grid line originally, you will not be able to move it to a grid line with this option selected. For example, if you set the grid to be 4x4 and have a control with a position of (1,1), when you move the control, it will only go to positions that are 4 units away in either dimension. Possible coordinates would be (5,5), (5,9), (9,5), and so on.
4. Select **Show grid** and choose **OK**.
5. Align controls by moving them around in the dialog box template. (You can move controls by dragging them or by selecting them and pressing the arrow keys.)

### Aligning Controls in Columns and Rows

The **Align→Array** command arranges controls in columns and rows, aligning them horizontally or vertically, spacing them evenly horizontally or vertically, and renumbering them so they are all in sequence.

To use the **Align→Array** command

1. Select the controls you want to arrange in columns and rows.

The selection frame surrounds the selected controls. The individual controls that will be affected by the **Array** command are indicated by shaded outlines.
2. If necessary, enlarge or reduce the size of the selection frame to enclose the area you want to fit the columns and rows into. For example, if you make the selection frame larger, the **Array** command will move the controls out to the new boundaries set by the selection frame.
3. Choose **Align→Array** or click the **Duplicate** tool.


## Working with Dialog Boxes

4. Under **Array layout**, specify the number of rows and columns you want.
5. Under **Order**, select how to order the controls in this group.
6. After you have chosen the options for the array, choose **OK**.

**Note:** For multiple selected controls, the **Duplicate** tool has the same effect as **Align→Array**. When only a single control is selected, it has the same effect as **Edit→Duplicate**.

### Undoing actions

You can "undo" any editing you do in the Dialog editor, such as placing controls, aligning them, deleting controls, and so on, with the **Undo** tool or the **Edit→Undo** command.

To specify the number of actions that you can undo, use the **File→Preferences** command. For more information,  see "Customizing Resource Workshop" on page 628.

## Setting the Windows Class of a Control

If you are an advanced Windows programmer, you may want to set the Windows class of a control.

For information about creating Windows classes, search Win32 online help (WIN32.HLP) for "Windows Classes."

To set the Windows class of a control,

1. Hold down Ctrl and double-click the control.
2. In the **Class** box, type the name of a Windows class.

**Note:** In the **Generic Control Style** window, you can also specify a **Caption**, **Control ID**, and **Style**. If you type anything next to **Info**, the dialog box will not be compatible with the Microsoft Resource Compiler.

## Setting Common Properties using the Properties Window



The standard controls share common properties that you can set using the **Properties** window.

To set a common property using the **Properties** window,

1. Select the control.
2. In the **Properties** window, select the property.
3. In the **Properties** window text box, type a value for the property.

**Note:** If the **Properties** window is not visible, choose **Options→Show properties**.

## Working with Dialog Boxes

Property	Description
<b>Caption</b>	Text to display on or next to the control. Different types of controls have captions in different areas. For example, in a group box, the caption is at the top left. In a push button, the caption is inside the button.
<b>Height</b>	The height of the control in pixels or dialog units
<b>ID source / Control ID</b>	The identifier for the control. For more information about identifiers,  see “Creating a Resource Script File” on page 620. For information about special identifiers for push button controls,  see “Push Button Properties” on page 646.
<b>ID value</b>	The unique number for the control
<b>Left</b>	The distance of the control from the left edge of its parent window in dialog units or pixels
<b>Tab stop</b>	Determines whether the user can use the Tab key to select the control
<b>Top</b>	The distance of the control from the top edge of its parent window in dialog units or pixels
<b>Visible</b>	Determines whether the control is visible
<b>Width</b>	The width of the control in dialog units or pixels

### Setting Common Properties using the Style Window

In addition to the common properties that you can set in the **Properties** window, each control has additional properties that you can set only in the **Style** window.

To set a common property using the **Style** window,

1. Double-click the control. The **Style** window appears.
2. In the **Attributes** group, set any of the properties described in the following table.

Property	Description
<b>Caption type</b>	Specifies whether the caption is text or a number
<b>Horizontal scroll bar</b>	Specifies whether the control has a horizontal scroll bar
<b>Vertical scroll bar</b>	Specifies whether the control has a vertical scroll bar

## Working with Dialog Boxes

<b>Alignment</b>	Specifies whether text is aligned to the right, left, or center of the control
<b>Owner drawing</b>	Determines the amount of control the application has over the appearance of the control. For more information about the <b>Owner drawing</b> property, choose the <b>Help</b> button on the <b>Style</b> window.
<b>Owner draw</b>	Determines whether the application handles the appearance of the control. When <b>Owner draw</b> is selected, the control sends a WM_DRAWITEM message to its parent.
<b>No character underline</b>	The <b>No character underline</b> check box turns off character underlining. You can underline a text character in your static control by preceding it with an ampersand (&). If you check <b>No character underline</b> , underlining is disabled and ampersands are displayed as literal characters.
<b>Border</b>	Specifies whether the control has a border

## Setting the Unique Properties of a Control

In addition to the common properties described in the previous two sections, each standard control has unique properties that you can set in the **Style** window. The unique properties of each control are described in the following sections.

**Push Button Properties** The push button control has unique properties for **Type** and **Control ID**.

**Type property:** You can set the **Type** property to one of the following values:

Value	Description
<b>Push button</b>	A normal push button
<b>Default push button</b>	Identical to a push button, but also includes a bold border indicating that it is the default response if the user presses Enter

**Control ID Property:** Just as with the other controls, you can set the **Control ID** property of a push button to any integer value. However, the following values have predefined meanings:

## Working with Dialog Boxes

Figure 175. Push Button - Control ID Property

Control ID	Value	Meaning
IDOK	1	OK
IDCANCEL	2	Cancel
IDABORT	3	Abort
IDRETRY	4	Retry
IDIGNORE	5	Ignore
IDYES	6	Yes
IDNO	7	No

**Radio Button Properties** The radio button control has a unique **Type** property that you can set to **Normal** or **Auto**. If you set the **Type** to **Auto**, Windows automatically selects or unselects the button; your application doesn't have to do the painting.

**Check Box Properties** The check box control has a unique **Type** property that you can set to one of the following values:

Value	Description
<b>Normal</b>	A normal check box. Your application is responsible for checking or unchecking the box.
<b>Auto</b>	A normal check box that Windows automatically checks or unchecks
<b>3 state</b>	A check box with three states: checked, unchecked, and dimmed. Your application is responsible for activating each state.
<b>Auto 3 state</b>	A 3 state check box that Windows automatically checks, unchecks, or dims

**Static Text Properties** The static text control has a unique **Type** property. You can set the **Type** property to one of the following values:

Value	Description
<b>Left text</b>	Displays text flush left within the control border (default). If text would extend past the edge of the frame, it automatically wraps to a new line.
<b>Left text--No wrap</b>	Displays text flush left within the control border. Any line of text that extends past the edge of the frame is clipped.

## Working with Dialog Boxes

<b>Centered text</b>	Displays text centered within the control border. If text would extend past the edge of the frame, it automatically wraps to a new line.
<b>Right text</b>	Displays text flush right within the control border. If text would extend past the edge of the frame, it automatically wraps to a new line.
<b>Simple text</b>	Displays a single line of flush-left text. Doesn't take tab characters and can't be broken to a new line. Simple text doesn't process the WM_CTLCOLOR message. Uses the current window background color.

**Note:** The text in all these styles uses the current Windows text color from the **Control Panel**. Also, in all static text but simple text, you can tab text by typing \T, and you can break text to a new line with \R.

### Scroll Bar Properties

The scroll bar control has a unique **Alignment** property that you can set to any of the following values:

<b>Option</b>	<b>Description</b>
<b>None</b>	The scroll bar fills the entire selection frame (default). If you resize the selection frame, you can change the scroll bar's proportions, making the arrow buttons and scroll box wider than usual.
<b>Top left</b>	A horizontal scroll bar displays at the top of the selection frame and extends the full width of the frame. A vertical scroll bar displays at the left side of the selection frame and extends the full height of the frame.
<b>Bottom right</b>	A horizontal scroll bar displays at the bottom of the selection frame and extends the full width of the frame. A vertical scroll bar displays at the right side of the selection frame and extends the full height of the frame.

### Edit Box Properties

The edit box control has several unique properties, which are described in the following sections.

**Case Property:** You can set the **Case** property to one of the following values:

<b>Value</b>	<b>Description</b>
<b>Case insensitive</b>	Displays text exactly as typed (default)
<b>Upper case</b>	Displays all text in uppercase letters, regardless of how it is typed

## Working with Dialog Boxes

**Lower case** Displays all text in lowercase letters, regardless of how it is typed

**Line Property:** You can set the **Line** property to one of the following values:

Value	Description
Single line	Limits the edit text to a single line (default)
Multiple line	Lets the user type text on multiple lines. (To enable scrolling of multiple-line text, set the <b>Vertical Automatic Scroll</b> option to on.)

**Miscellaneous Boolean Properties:** You can turn the following properties on or off:

Property	Description
----------	-------------

**Automatic horizontal scroll**

When the user types a character at the right edge of the edit text boundary, the text automatically scrolls ten characters to the right. When the user presses Enter, the text scrolls back to the zero position.

**Automatic vertical scroll**

When the user presses Enter on the last line of the edit text control, the text scrolls up a full page. For example, if the control is five lines long, pressing Enter on the last line causes text to scroll up five lines; the cursor goes back to the top line.

**Password**

When Password is on, the letters being typed are not displayed. Instead, asterisks appear in their place. This is helpful for keeping passwords secret.

**Convert OEM**

Text typed into the control is converted to the current OEM character set, then reconverted to ANSI. This option is useful in file input boxes because it ensures that any file name entered can be translated into the OEM character set, which is used by the DOS file system.

**Keep selection**

Keeps selected text highlighted, even when this control doesn't have keyboard focus. For example, if a user highlights text in an edit text control and then moves to another control, the text will no longer be highlighted, unless the edit text control has the **Keep selection** property turned on.

## Working with Dialog Boxes

### List Box Properties

The list box control has the following unique properties, which you can turn on or off:

Property	Description
<b>Notify</b>	Sends an input message to the parent window when the user clicks an item in the list (default)
<b>Sort</b>	Sorts the list alphabetically
<b>Multiple select</b>	Lets the user select more than one item at a time. The user can also toggle individual items on and off.
<b>Don't redraw</b>	Prevents the list box from being redrawn when it is changed
<b>Tab stops</b>	Organizes the information in the list box in columns. The default column width is 32 dialog units or 8 characters. You should use tab characters. (If you want to change the column width, the application should set its own tab stops using the LB_SETTABSTOPS message.)
<b>Integral height</b>	Decreases list box height at run time, if necessary, to the nearest integral multiple of the current font height (default). For example, a list box might have three items that display completely, but a fourth that is partially cut off. If <b>Integral height</b> is turned on, the list box decreases its size at run time to the space required for three items (three times the font height).
<b>Multi column</b>	Creates a list box in which the text wraps from column to column. The user scrolls the list box horizontally to display additional text. If you turn this option on, the application must send the LB_SETCOLUMNWIDTH message to set the width of all columns in pixels.
<b>Pass keyboard input</b>	Passes what the user types to the application
<b>Extend select (used with multiple-select boxes)</b>	Lets the user select more than one item in a list
<b>Has strings</b>	Specifies list box contents. If you have chosen either the <b>Fixed</b> or <b>Variable owner drawing</b> option, the list box stores text for each list item with the LB_INSERTSTRING or LB_ADDSTRING message. The list box can also retrieve list items from the message LB_GETTEXT.

### Combo Box Properties

The combo box control has several unique properties, which are described in the following sections.



## Working with Dialog Boxes

**Type Property:** You can set the **Type** property to one of the following values:


Value	Description
<b>Simple</b>	The drop-down list is always expanded to display items in the list, and the user can edit the items in the list (default)
<b>Drop down</b>	When the dialog box is first displayed, the combo box consists of a single line of editable text. The user can click the down arrow to expand the list, and edit all items in the list.
<b>Drop down list</b>	This option works like a drop down, but the list is static. The user can select, but can't change anything in the list.

**Miscellaneous Boolean Properties:** You can turn the following properties on or off:

Property	Description
<b>Sorted</b>	Automatically sorts items in a list box in alphabetical order
<b>Integral height</b>	Sizes the list box at run time so all items in the list are completely displayed (default). If you want to control the height of the list box precisely, uncheck this option.
<b>OEM conversion</b>	Converts text the user types in to the current OEM character set, then reconverts the text to ANSI. This option is useful in file input boxes because it ensures that any file name entered can be translated into the OEM character set, which is used by the DOS file system.
<b>Auto horizontal</b>	Scrolls text to the left automatically when it exceeds the width of the control

### Icon Properties

To place an iconic static control in your dialog box,

1. In the **Tools** palette, click the **Icon** tool.
2. In the dialog box template, drag where you want the icon to appear.  
The icon control appears in the dialog box template.
3. Double-click the icon control.
4. In the **Caption** box, type the identifier for an icon resource. For information about creating an icon resource,  see Chapter 60, “Working with Graphics” on page 673.
5. Select the **Number** option.
6. Choose **OK**.  
The icon appears in your dialog box.

## Working with Dialog Boxes

### Frame Properties

The frame control has a unique **Type** property, which you can set to any of the following values:

Value	Description
<b>White frame</b>	Displays an empty frame with a solid outline that uses the current Windows background color set in the <b>Control Panel</b> . In Windows, the default color for the background is white.
<b>Gray frame</b>	Displays an empty frame with a solid outline that uses the current screen background (desktop) color set in the <b>Control Panel</b> . In Windows, the default color for the desktop is gray.
<b>Black frame</b>	Displays an empty frame with a solid outline that uses the current Windows frame color set in the <b>Control Panel</b> . The Windows default color for window frames is black.

### Rectangle Properties

The rectangle control has a unique **Type** property, which you can set to any of the following values:

Value	Description
<b>White rectangle</b>	Displays a filled rectangle that uses the current Windows background color set in the <b>Control Panel</b> . In Windows, the default color for the background is white.
<b>Gray rectangle</b>	Displays a filled rectangle that uses the current screen background (desktop) color set in the <b>Control Panel</b> . In Windows, the default color for the desktop is gray.
<b>Black rectangle</b>	Displays a filled rectangle that uses the current Windows frame color set in the <b>Control Panel</b> . In Windows, the default color for window frames is black.

### Line Properties

The line control has a unique **Type** property, which you can set to any of the following values:

- **Horizontal dip**
- **Vertical dip**
- **Horizontal bump**
- **Vertical bump**

A dip appears to be etched into the surface of the dialog box; a bump appears to be raised above the surface of the dialog box.

---

### Testing a Dialog Box Resource

To see the effect of any changes you have made to the dialog box resource, select the **Test** tool or choose **Options→Test dialog**.

You can press Tab and the arrow keys to see how you can move around your dialog box, or you can type text to see how text is scrolled in an edit text control. Check to see if your controls are in the order you want them.

When you test a dialog box, the status line at the bottom of the dialog editor says "Test".

### Testing Two Dialog Boxes at the Same Time

To test two dialog boxes at the same time:

1. In the **Resource Project** window, double-click on the name of the first dialog box. The Dialog editor starts and displays that dialog box.
2. Click the **Test** tool or choose **Options→Test dialog**.
3. Click the **Minimize** button of the dialog editor twice. (The first click switches focus from the dialog box to the **Dialog editor** window.) Your dialog box is now floating and modeless; you can move it like any other window.
4. Return to the **Resource Project** window and double-click the name of the second dialog box you want to view. A second dialog editor starts up.
5. Again, click the **Test** tool or choose **Options→Test dialog** in the second dialog editor.
6. Click the **Minimize** button of the second dialog editor twice.

Now you have two floating dialog boxes that you can test at the same time.

### Returning to Edit Mode

To leave test mode and return to edit mode, do one of the following:

- In the dialog box, choose **OK** or **Cancel**.
- Choose **Options→Test dialog** again.
- Press Enter.
- Click the **Selector** twice. (The first click switches focus from the dialog box to the dialog editor.)

## Working with Dialog Boxes

---

### Customizing the Dialog Editor

You can customize the dialog editor by choosing **Options→Preferences** and specifying any of the following options:

Option	Settings and Descriptions
<b>Status line units</b>	<p>Determines the unit of measurement the status line uses to display information:</p> <p><i>Dialog.</i> Uses the dialog unit as the unit of measurement on the status line. In a dialog unit, <math>y</math> equals <math>1/8</math> of the font height, and <math>x</math> equals <math>1/4</math> of the font width.</p> <p><i>Screen.</i> Uses pixels as the unit of measurement on the status line.</p>
<b>Selection border</b>	<p>Lets you change the appearance of the frame that surrounds selected controls:</p> <p><i>Thick frame.</i> The selection frame is thick, like the standard frame around a Windows application or dialog box window (default).</p> <p><i>Handles.</i> The selection frame is a rectangular outline with handles (small squares) at each corner and at the midpoint of each side.</p>
<b>Drawing type</b>	<p>Determines how elements of your dialog box are displayed in the dialog editor:</p> <p><i>Draft.</i> Draws each control as a rectangle with its control ID in the center. This option also lets you see how much space is occupied by the control's selection frame.</p> <p><i>Normal.</i> Draws standard Windows controls as they will appear at run time. Drawing of custom controls is determined by the <b>Draw custom controls as frames</b> check box, described shortly.</p> <p><i>WYSIWYG</i> (the default option). With this option selected, Resource Workshop creates the dialog and control child windows and the controls draw themselves. This option is slowest, but the most accurate. Installable custom controls draw themselves.</p>
<b>Selection rules</b>	<p>Determines how controls are selected. If you are working with closely spaced controls, use these options for greater precision and to avoid selecting controls inadvertently:</p>

## Working with Dialog Boxes

*Select Near Border.* If this option is checked, you must click *on* the control's border. If it is not checked, you can click anywhere inside the control's border.

*Selection Rectangle Surrounds.* If this option is checked, you must entirely surround the control (or controls) with the selection rectangle. If it is not checked, the selection rectangle need only touch the control (or controls).

In the resource script language, each type of control has a unique syntax. For example, centered static text uses the CTEXT statement. The CONTROL statement, however, can specify any type of control. If you want to generate only CONTROL statements in your resource script (rather than the specialized control statements), select the **Generate CONTROL statements only** option. For more information about the resource script language, search help for "Editing a Resource as Text."

The **Draw custom controls as frames** check box is available only when the **Drawing type** is set to **Normal**. When the option is checked, custom controls are drawn as empty rectangular outlines. When the option is unchecked, custom controls are drawn as gray rectangles with their text (if any) in a white rectangle in the center. Drawing custom controls as frames can speed up drawing of your dialog boxes on the screen.

If you check CTL3D2V.DLL, the dialog editor uses the Windows 3-D look for controls such as radio buttons and check boxes.

---

## Using a Dialog Box Resource in an Application

You can program a dialog box with the Windows API.

### Programming a Dialog Box with the Windows API

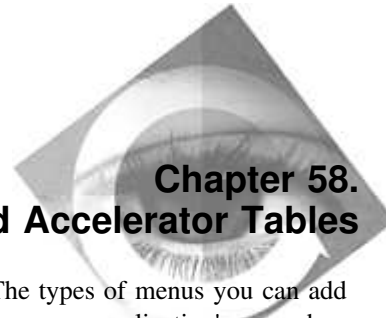


To program a dialog box with the Windows API, use the *DialogBox* function. For example, the following code shows how to create and display a dialog box.

```
HINSTANCE hInst;  
HWND hwndParent;  
DLGPROC dlgProc;  
dlgProc = (DLGPROC) MakeProcInstance(ResModDlgProc, hInst);  
DialogBox(hInst, MAKEINTRESOURCE(IDD_RESDEMODIALOG), hwndParent, dlgProc);  
FreeProcInstance((FARPROC) dlgProc);
```

For more information about programming dialog boxes with the Windows API, search the Win32 Online Help (win32.hlp) file for "Dialog Boxes."

## **Working with Dialog Boxes**



## Chapter 58.

# Working with Menus and Accelerator Tables

A *menu* is a list of actions that a user can perform. The types of menus you can add to an application include standard menus, which appear on an application's menu bar; floating menus, which can appear anywhere on the screen; and hierarchical (cascaded) menus, which appear when you choose a menu item. Applications often have standard menus that are labeled File, Edit, and Help; these menus contain file manipulation, editing, and online help commands, respectively.

A *menuex* is an extended menu. It provides the same functionality as a menu, with the following extensions: help IDS on popup menus, IDs on popup menus, and the use of the menu type and state flags.

An *accelerator table* contains one or more combination of keys that a user can press to perform an action. For example, many applications use Ctrl+X to cut data and Ctrl+V to paste data.

Like other resources, you define a menu, menuex, or accelerator with a resource script. This chapter shows how to edit a resource script for a menu, menuex, or accelerator table, assign an accelerator to a menu item, and use the corresponding resources in an application. This chapter also shows how to customize the menu and accelerator editors.

---

## Editing a Menu Resource Script

You edit menu resources by using the menu editor.

To start the menu editor:

1. Add a new menu resource script to a resource script file. Or open a resource script file that contains a menu resource script.
2. If the menu editor doesn't start automatically, double-click a menu in the **Resource Project** window.

The menu editor appears. It contains three panes:

- The **Attribute** pane is where you edit existing menus and menu commands.
- The **Test** pane displays your menu and lets you test it.
- The **Outline** pane shows the menus, menu items, and separators in pseudocode. To see the complete code with all parameters for each statement, edit the

## Working with Menus and Accelerator Tables

resource script file with a text editor. For more information, search online help for "Editing a Resource as Text".

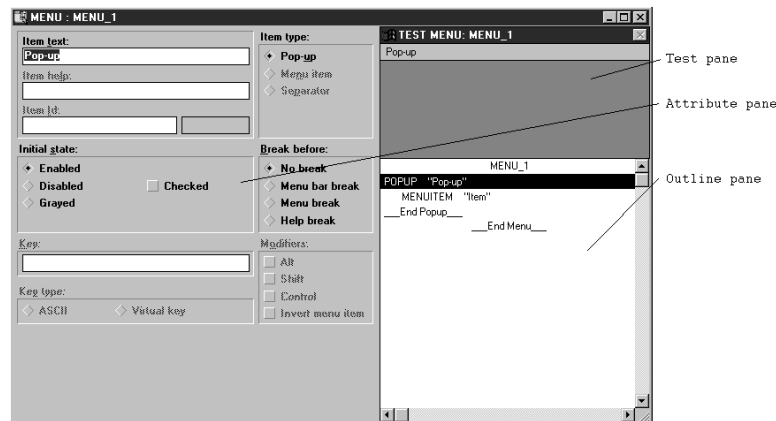



Figure 176. Menu editor

After you start the menu editor, you can edit the menu resource script by:

- Adding a menu or separator to the menu bar
- Adding a menu item or separator to a menu
- Setting the initial state of a menu or menu item
- Changing the identifier for a menu item
- Associating help text with a menu item
- Assigning an accelerator to a menu item
- Copying, moving, or removing a menu, menu item, or separator
- Checking for duplicate identifiers

To customize the menu editor,  see “Customizing the Menu Editor” on page 664.

### Adding a Menu or Separator to the Menu Bar

You can add a predefined menu, custom menu, or separator to the menu bar. With a separator, you can place menus to the far right or on different lines of the menu bar.

#### Adding a Predefined Menu

To add a predefined menu to the menu bar:

1. In the **Outline** pane, select a line between the POPUP and End Popup lines of a menu; this menu will immediately follow the predefined menu. (You can't add a predefined menu as a cascaded menu.)
2. Choose one of the following commands:



## Working with Menus and Accelerator Tables

Command	Description
<b>Menu→New file pop-up</b>	Adds a <b>File</b> menu that contains the following commands: <b>New</b> , <b>Open</b> , <b>Save</b> , <b>Save as</b> , <b>Print</b> , <b>Page setup</b> , <b>Printer setup</b> , and <b>Exit</b>
<b>Menu→New edit pop-up</b>	Adds an <b>Edit</b> menu that contains the following commands: <b>Undo</b> , <b>Cut</b> , <b>Copy</b> , and <b>Paste</b>
<b>Menu→New help pop-up</b>	Adds a <b>Help</b> menu that contains the following commands: <b>Index</b> , <b>Keyboard</b> , <b>Commands</b> , <b>Procedures</b> , <b>Using help</b> , and <b>About</b>

### Adding a Custom Menu

To add a custom menu to the menu bar:

1. In the **Outline** pane, select the line that will immediately precede the menu. If you select a line inside an existing menu (between the POPUP and End Popup lines), the new menu becomes a cascaded menu.
2. Choose **Menu→New pop-up**. Or press Ctrl+P.
3. In the **Attributes** pane, type the menu name in the **Item text** box.

To associate the menu with a shortcut key, include an ampersand (&) in the menu name. The letter following the ampersand will be underlined, and the user will be able to select the menu by pressing Alt plus the underlined letter. If you want an ampersand character to appear in the menu name, use two ampersands (&&).

You can also use the tab character (\t) and the right-align character (\a) in the menu name.

Now you can test your menu in the **Test** pane.

### Adding a Separator

To add a separator to the menu bar:

1. In the **Outline** pane, select the POPUP line of a menu; this menu that will immediately follow the separator.
2. In the **Attribute** pane, choose one of the following options:

Option	Description
<b>Menu bar break or Menu break</b>	Starts a new line in the menu bar
<b>Help break</b>	Moves the menu to the far right of the menu bar

## Working with Menus and Accelerator Tables

### Adding a Menu Item or Separator to a Menu

To add a menu item to a menu:

1. In the **Outline** pane, select the menu item that will immediately precede the new menu item.
2. Choose **Menu→New menu item**, or press Ins. .

The new menu item appears.

3. In the **Attribute** pane, type the name of the new menu item in the **Item** box.

To associate the menu item with a shortcut key, include an ampersand (&) in the menu item name. The letter following the ampersand will be underlined, and the user will be able to select the menu item by pressing Alt plus the underlined letter. If you want an ampersand character to appear in the menu item name, use two ampersands (&&).

You can also use the tab character (\t) and the right-align character (\a) in the menu item name.

To add a vertical separator to a menu:

1. In the **Outline** pane, select the menu item that will immediately follow the separator.
2. In the **Attribute** pane, choose one of the following options:

Option	Description
<b>Menu bar break</b>	Starts a new column that is preceded by a vertical line in the menu
<b>Menu break</b>	Starts a new column in the menu

To add a horizontal separator to a menu:

1. Select the menu item that will immediately follow the separator.
2. Choose **Menu→New menu item**, or press Ins.
3. In the **Attribute** pane, choose **Separator** in the **Item type** group.

### Setting the Initial State of a Menu or Menu Item

To set the initial state of a menu or menu item:


1. In the **Outline** pane, select the menu or menu item.
2. In the **Attribute** pane, choose one of the following options:

## Working with Menus and Accelerator Tables

Option	Description
<b>Enabled</b>	In its initial state, the menu or menu item is enabled. Use the <code>EnableMenuItem</code> function to change the state of the menu or menu item in code.
<b>Disabled</b>	In its initial state, the menu or menu item is disabled. The user can't distinguish between Enabled and Disabled commands; they look the same on the menu. Use the <code>EnableMenuItem</code> function to change the state of the menu item.
<b>Grayed</b>	In its initial state, the menu or menu item is disabled and its text is grayed. The shading lets the user know the command is not currently available. Use the <code>EnableMenuItem</code> function to change the state of the menu or menu item.

3. In the **Attribute** pane, select or unselect the **Checked** box. If you select the **Checked** box, a check mark appears next to the menu item. You can use a check mark to represent an option that can be turned on or off. You can check or uncheck the menu item in code with the `CheckMenuItem` function.

### Changing the Identifier for a Menu Item

An identifier is automatically assigned to each menu item that you create. You can change the identifier for the current menu item by typing a name or value in the **Item ID** box, or by using the **Resource→Identifiers** command. For more information about creating and editing identifiers,  see “Creating a Resource Script File” on page 620.

### Associating Help Text with a Menu Item

You might want to associate Help text with a menu item so that you can easily display information about the menu item in the status bar of an application.

To associate help text with a menu item:

1. In the **Outline** pane, select the menu item.
2. In the **Attributes** box, type the text into the **Item help** box.

The help text is automatically added to a string table in the current resource script file. (If one doesn't already exist, a string table is automatically added to the current resource script file.)

## Working with Menus and Accelerator Tables

### Assigning an Accelerator to a Menu Item

You can assign an accelerator to a menu item in Key Value or Manual mode.

In Key Value mode, any key or key combination you press is automatically assigned to the menu item. You don't have to manually type the key name, select a key type (ASCII or virtual), or select combination keys (Alt, Shift, and Ctrl).

In Manual mode, you provide all the information that defines the accelerator. You must decide whether to use an ASCII or virtual key. (In general, an ASCII key is an alphanumeric character or punctuation mark, and a virtual key is a function key, arrow key, or an editing key, such as *Home* or *PgDn*.) If it is a virtual key, you must type the correct identifier for the key as it is defined in the `windows.h` file. You must also select the appropriate **Key type** radio button (ASCII or Virtual Key) and select the appropriate combination of the **Alt**, **Shift**, and **Ctrl** check boxes.

To assign an accelerator to a menu item in Key Value mode:

1. In the **Outline** pane, select the menu item.
2. Choose **Menu→Accelerator key value**.
3. Press the keyboard combination for the accelerator.
4. Press Esc.
5. Select the **Invert** option if you want the corresponding menu to flash when the user presses the accelerator key.

To assign an accelerator to a menu item in Manual mode:

1. In the **Outline** pane, select the menu item.
2. In the **Key** box in the **Attribute** pane, specify an ASCII or virtual key.


To specify an ASCII key, type the key in quotation marks. Typically you don't use a single ASCII character as an accelerator key; instead you combine the character with the Alt or Ctrl keys. To add the Ctrl key, precede the ASCII key with a caret (^). For example, type "`^P`" to assign Ctrl+P to the menu item. To add the Alt key, select the **Alt** option in step 4.

To specify a virtual key, type the identifier for the key in uppercase letters. The identifiers for virtual keys all start with `VK_` and are defined in the `windows.h` file. You can combine the virtual key with the Alt, Shift, and Ctrl keys in step 4.

3. In the **Key type** group, choose **ASCII** or **Virtual key**, depending on the key you specified in step 2.
4. Select any of the following options:

## Working with Menus and Accelerator Tables

Option	Description
<b>Alt</b>	Combines the Alt key with the accelerator key (for example, Alt+W)
<b>Shift</b>	Combines the Shift key with the accelerator key (for example, Shift+F1). You can't use the Shift option with an ASCII key.
<b>Control</b>	Combines the Ctrl key with the accelerator key (for example, Ctrl+W). You can't use the Control option with an ASCII key. However, you can combine the Ctrl key with an ASCII key by including a caret (^) in the key name. (See step 2.)
<b>Insert</b>	Causes the corresponding menu to flash when the user presses the accelerator key combination

The accelerator is automatically added to an accelerator table in the current resource script file. For more information about accelerator tables,  see “Customizing the Menu Editor” on page 664.

### Copying, Moving, or Removing a Menu, Menu Item, or Separator

You can use the commands on the **Edit** menu to copy, move, or remove the current menu, menu item, or horizontal separator.

**Note:** You can't remove the last menu, menu item, or horizontal separator from a menu. However, you can remove the menu resource script by selecting the script in the **Resource Project** window and pressing Del.

You can't copy or move a vertical separator, but you can remove it.

To remove a vertical separator:

1. In the **Outline** pane, select the menu or menu item that follows the separator.
2. In the **Attribute** pane, choose **No break**.

### Checking for Duplicate Identifiers

To check for duplicate identifiers, choose **Menu→Check duplicates**.

If two menu items have the same identifier, the message "Duplicate command value found" appears, and the second menu item with a duplicate identifier appears in the **Attribute** pane.


## Working with Menus and Accelerator Tables

---

### Creating a Resource Script for a Floating Menu

A floating menu can be displayed anywhere on the screen; it is not tied to a menu bar.

To create a resource script for a floating menu:

1. Using the **File→Add to resource project** command, add a menu resource indirectly to the current resource script file. For more information,  see “Adding a New Resource Script Indirectly” on page 621.
2. In the **Outline** pane, select the name of the resource. (The name is centered on the first line.)
3. Press the Ins key to add at least one menu item at the top of the outline.
4. Remove the default menu.
5. Add, edit, or remove menu items. These procedures are described in the previous sections.
6. Using the **Resource→Save resource as** command, save the menu resource script in a new resource script file.

**Note:** To see how the resource will appear at run time, choose **View→View as Pop-up**. When you view the menu in the **Test Menu** pane, it will still appear tied to the menu bar, but as long as your code uses the `TrackPopupMenu` function correctly, the menu will float at run time.

---

### Customizing the Menu Editor

You can use the following commands to customize the menu editor:

Option	Description
<b>View→View as Pop-up</b>	Select this option to have menus displayed as cascaded menus in a single main menu; deselect this option to have menus displayed on the menu bar.  Leave <b>View as Pop-up</b> unchecked if your menu resource contains the application's entire menu structure and you want it displayed as it would appear to the user.  If you're working on a floating menu, check this option to display the test menu as it actually would appear. The command <b>Pop-up</b> appears on the menu bar, and you select <b>Pop-up</b> to display the menu itself.

## Working with Menus and Accelerator Tables

<b>View→First graphic</b>	This graphic represents the default configuration of the panes, with the <b>Test Menu</b> pane over the <b>Outline</b> pane and to the right of the <b>Attribute</b> pane.
<b>View→Second graphic</b>	Check this graphic to put the <b>Test Menu</b> pane across the top of the edit window, like a normal menu bar.
<b>Menu→Change identifier prefix</b>	Use this command to change the identifier prefix that is automatically assigned to new menu items.
<b>Menu→Track test menu</b>	Select this option if you want menu items to appear in the <b>Attribute</b> pane when you select them in the <b>Test</b> pane.

---

### Editing a Menuex Resource Script

You edit menuex resources using the text editor.

To start the text editor:

1. Add a new menuex resource script to a resource script file. Or open a resource script file that contains a menuex resource script.
2. If the text editor doesn't start automatically, double-click a menuex in the **Resource Project** window.

The text editor appears. For more information about the syntax of the menuex resource, search the Win32 Online help (win32.hlp) file for "Menuex Resource".

---

### Editing an Accelerator Table Resource Script

You can create accelerators for menu items using the menu or accelerator editors. However, you must use the Accelerator editor to remove accelerators and to create accelerators that are not linked to menu items.

To start the accelerator editor:

1. Add a new resource script for an accelerator table to a resource script file, or open a resource script file that contains a resource script for an accelerator table.
2. If the accelerator editor doesn't start automatically, double-click an accelerator table in the **Resource Project** window.

The accelerator editor appears. It contains two panes:

- The **Attribute** pane is where you edit existing accelerators.

## Working with Menus and Accelerator Tables

- The **Outline** pane shows you, in outline script form, all the accelerators defined in the table. The top line of this outline is the name of the accelerator table. The lines below it are accelerator entries, which have three parts: the keyboard combination, the identifier for the accelerator, and virtual or ASCII indicator.

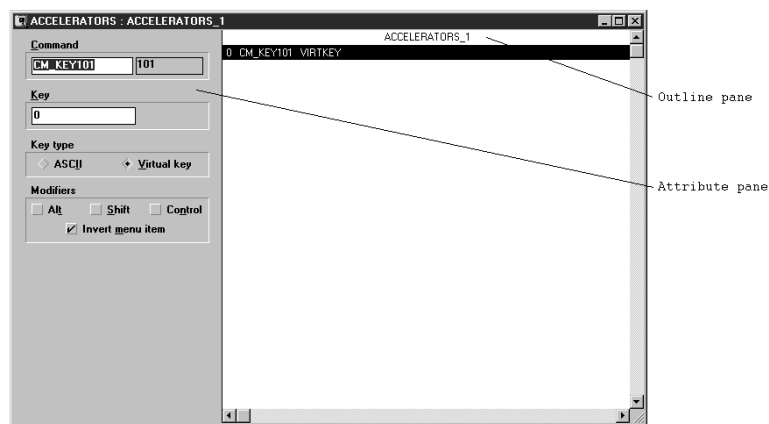


Figure 177. Accelerator editor

After you start the accelerator editor, you can edit the resource script for the accelerator table by:

- Adding an accelerator resource
- Changing the identifier for an accelerator resource
- Changing the keyboard combination for an accelerator resource
- Copying, moving, or removing an accelerator resource
- Checking for duplicate keyboard combinations

### Adding an accelerator resource

To add an accelerator resource to the accelerator table,

1. In the **Outline** pane, select the accelerator resource that will immediately precede the new accelerator resource.
2. Choose **Accelerator→New item**, or press **Ins**.


When you add an accelerator resource, the editing focus automatically switches to the **Attribute** pane.

### Changing the Identifier for an Accelerator Resource

An identifier is automatically assigned to each accelerator resource that you create. In the **Attribute** pane, you can change the identifier for the current accelerator resource by typing a name or value in the **Command** box, or by using the



## Working with Menus and Accelerator Tables

**Resource→Identifiers** command. For more information about creating and editing identifiers,  see “Creating a Resource Script File” on page 620.

If you type the identifier for a menu item in the **Command** box, the accelerator is assigned to the menu item.


### Changing the Keyboard Combination for an Accelerator Resource


You can change the keyboard combination for an accelerator resource in Key Value or Manual mode. These modes are described in “Assigning an Accelerator to a Menu Item” on page 662.

To change a keyboard combination for an accelerator resource in Key Value mode,

1. In the **Outline** pane, select the accelerator resource. You can use the mouse or keyboard (Ctrl+↑ or Ctrl+↓).
2. Choose **Accelerator→Key value**.
3. Press the new keyboard combination for the accelerator.
4. Select the **Invert** option if you want the corresponding menu to flash when the user presses the accelerator key. If the accelerator key is not assigned to a menu item, then the **Invert** option is ignored.

To change the keyboard combination for an accelerator resource in Manual mode,

1. In the **Outline** pane, select the accelerator resource. You can use the mouse or keyboard (Ctrl ↑ or Ctrl ↓).
2. In the **Command** box in the **Attribute** pane, type an ASCII or virtual key.  
For more information about ASCII and virtual keys,  see “Assigning an Accelerator to a Menu Item” on page 662.
3. In the **Key type** group, choose ASCII or Virtual Key, depending on the key you specified in step 2.
4. In the **Modifiers** group, select or deselect the **Alt**, **Shift**, **Control**, and **Invert** options.

For more information about these options,  see “Assigning an Accelerator to a Menu Item” on page 662.

### Copying, Moving, or Removing an Accelerator Resource

You can use the commands on the **Edit** menu to copy, move, or remove the current accelerator resource.

**Note:** You can't remove the last accelerator resource from an accelerator table. However, you can remove the resource script for the accelerator table by selecting the script in the **Resource Project** window and pressing Del.

## Working with Menus and Accelerator Tables

### Checking for Duplicate Keyboard Combinations

To check for duplicate keyboard combinations, choose **Accelerator→Check dup keys**.

If two accelerator resources use the same keyboard combination, the message "Duplicate key value found" appears, and the second of the accelerator resources with duplicate keyboard combinations is highlighted in the **Outline** pane.

---

### Customizing the Accelerator Editor

You can customize the accelerator editor by changing the identifier prefix that is automatically assigned to new accelerators.

To change the identifier prefix that is automatically assigned to new accelerators:

1. Choose **Accelerator→Change identifier prefix**.
2. Type the new identifier prefix.
3. Choose **OK**.

---

### Using Menu and Accelerator Resources in an Application

You can program menus and accelerators with the Windows API.

#### Programming menus and accelerators with the Windows API



To program menus and accelerators with the Windows API, use the `LoadMenu` and `LoadAccelerators` functions. For example, the following code shows how to associate a menu and accelerators with the main window.

```
HMENU hMenu;  
hMenu = LoadMenu(hInst, MAKEINTRESOURCE(IDD_MENU1)  
LoadAccelerators(hInst, MAKEINTRESOURCE(IDD_ACCELERATORS1))
```

In the preceding code, `IDM_MENU1` and `IDD_ACCELERATORS1` are the identifiers for the menu and accelerator table respectively.



## Chapter 59. Working with String Tables

A *string table* contains descriptions, prompts, error messages, and other text that your application displays.

Like other resources, you define a string table with a resource script. This chapter shows how to edit a string table resource script, and use the corresponding resource in an application. This chapter also shows how to customize the string editor.

---

### Editing a String Table Resource Script

You edit a string table resource script by using the string editor.

To start the string editor:

1. Add a new string table resource script to a resource script file, or open a resource script file that contains a string table resource script.
2. If the string editor doesn't start automatically, double-click a string table in the **Resource Project** window.

The string editor appears. It is similar to a three-column spreadsheet.

After you start the string editor, you can edit the string resource script by:

- Adding a string resource
- Changing the characters of a string resource
- Changing the identifier for a string resource
- Copying, moving, or removing a string resource

To customize the string editor,  see “Customizing the String Editor” on page 671.

### Adding a String Resource

To add a string resource:

1. Select the line containing the string resource that will immediately precede the new string resource.
2. Choose **Stringtable→New item**, or press Ins.
3. In the **String** column, type up to 255 characters. You can also use any of the C-type escape sequences, including \n (newline), \t (tab), \r (carriage return), \\ (backslash), and \" (double quote).
4. Press Enter.

## Working with String Tables

### Changing the Characters of a String Resource

To change the characters of a string resource:


1. In the **String** column, select the characters.
2. Type the new characters. You can also use any of the C-type escape sequences, including: \n (newline), \t (tab), \r (carriage return), \\ (backslash), and \" (double quote).
3. Press Enter.

### Changing the Identifier for a String Resource

To change the identifier for a string resource:

1. In the **ID Source** column, select the identifier.
2. Type a new identifier.
3. Press Enter.

**Note:** You can't directly change the identifier in the **ID Value** column; however, any integer that you type into the **ID Source** column is automatically copied into the **ID Value** column.

You can also change the identifier for a string resource by using the **Resource→Identifiers** command. For more information about identifiers,  see “Working with Resource Script Files” on page 620.

### Grouping String Resources to Maximize Memory Efficiency

String resources are linked to an application in 16-resource segments. String resources with identifier values from 0 to 15 are in the first segment, string resources with identifier values from 16 to 31 are in the second segment, and so on.

When an application uses a string resource, it loads all the string resources in the corresponding segment. Therefore, you can maximize the memory efficiency of an application by grouping related string resources together.

For example, suppose you have created six string resources for the title screen of an application. If you arrange these string resources into six different segments, the application could load over 24000 bytes of unused string resources, depending on the sizes of the string resources in each segment, when the title screen is displayed. On the other hand, if you group the six string resources into one segment, the application will load no more than 4100 bytes of string resources when the title screen is displayed.

## Working with String Tables

### Copying, Moving, or Removing a String Resource

You can use the commands on the **Edit** menu to copy, move, or remove a string resource. You can also perform these actions with the shortcut menu. To display the shortcut menu, right-click on a string resource.

**Note:** You remove all but the last string from a string table. You can remove the string table resource script by selecting it in the **Resource Project** window and pressing Del.

---

### Customizing the String Editor

You can customize the string editor by changing the identifier prefix that is automatically assigned to new string resources.

To change the identifier prefix that is automatically assigned to new string resources:

1. Choose **Stringtable→Change identifier prefix**.
2. Type the new identifier prefix.
3. Choose **OK**.

---

### Using a String Resource in an Application



To include a string resource in an application, use the *LoadString* function. For example, the following code shows how to copy a string resource into a variable.

```
char messageText[255];  
LoadString(IDS_STRING1, messageText, sizeof(messageText));
```

In the above code, IDS\_STRING1 is the identifier for the string.

## Working with String Tables



## Chapter 60. Working with Graphics

With Resource Workshop, you can create and edit the following types of graphics:

- A *bitmap*, which is a graphic image, such as a picture, logo, or other drawing.
- An *icon*, which is a graphic image that represents a minimized window. An icon is 64x64, 32x32, 16x32, or 16x16 pixels in size.
- A *cursor*, which is a graphic image that shows the position of the mouse on the screen and indicates the types of actions that a user can perform. A cursor is 32x32 pixels in size. Many applications use a pointer, which indicates that the user can make a selection, and an hourglass, which indicates that the system is performing an action.
- A *font*, which is a set of characters with a specific typeface and size. For example, **10-point Times Roman bold** is a font. A font can also contain a set of bitmaps that are not characters.

Like other resources, you define a graphic with a resource script. This chapter describes how to edit a resource script for a graphic, and use the corresponding resource in an application.

---

### Editing a Resource Script for a Graphic

You edit a resource script for a graphic by using the graphic editor.

To start the graphic editor:

1. Add a new resource script for a graphic to a resource script file.  
Or open a resource script file that contains a resource script for a graphic.  
Or create or open a graphic file.
2. If the graphic editor doesn't start automatically, double-click a graphic in the **Resource Project** window.
3. If you are editing a resource script for an icon or cursor that contains more than one image, double-click an image in the **Image** window. (The **Image** window has the identifier for the icon or cursor in its caption.)

The graphic editor appears. It contains a **Tools** palette and **Colors** palette.

## Working with Graphics

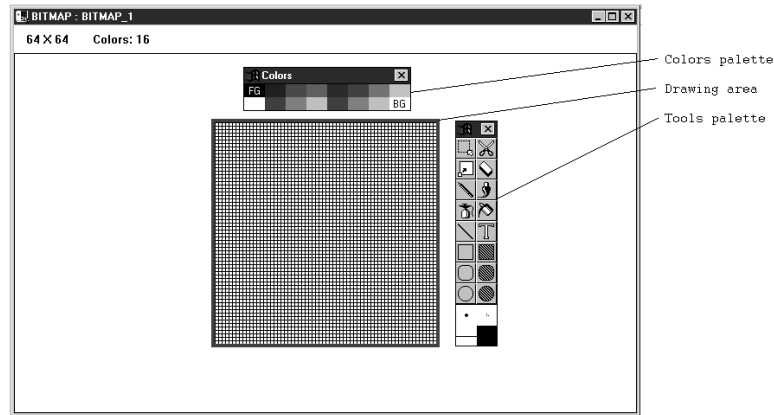


Figure 178. Graphic editor

The **Tools** palette contains the drawing tools.

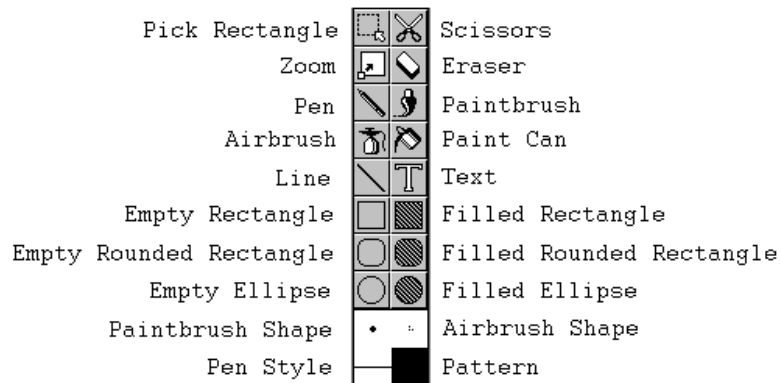



Figure 179. Graphic editor - Tools palette

The **Colors** palette shows the available colors.


To customize the graphic editor,  see “Customizing the Graphic editor” on page 686.

### Choosing Colors

You can draw or paint with up to 256 different colors, which you can choose from a palette of over 16 million colors. However, the maximum number of colors that you can use at a given time depends on the type of graphic that you are editing and the capabilities of your display device and driver. For example, if you are editing a cursor or font, you can use black and white only. If you are editing a bitmap or icon, the



## Working with Graphics

maximum number of colors you can use is specified in the **Colors** property of the bitmap or icon. For information about changing the **Colors** property of a bitmap or icon,  see the sections “Changing the Properties of a Bitmap” on page 682 and “Changing the Properties of an Icon” on page 683.

To view the capabilities of your display device and driver:

1. Edit a bitmap or icon.
2. Choose **Bitmap**→**Size and attributes** or **Icon**→**Size and attributes**.
3. Choose **Device Info**.

You can choose a foreground and background color. The *foreground* color is the color you draw with mouse button one; the *background* color is the color you draw with mouse button two.

To select the foreground color, click mouse button one on a color in the **Colors** palette.

To select the background color, click mouse button two on a color in the **Colors** palette.

In the **Colors** palette, the current foreground and background colors are denoted by the letters FG and BG respectively. The letters FB denote a color that is both the foreground *and* background color.

### Choosing a Color that Represents a Transparent or Inverted Area

When you are editing an icon or cursor, you can choose the colors that represent transparent and inverted areas.

- A *transparent* area “drops out” at run time, allowing the color behind the icon or cursor to show through. Transparency is especially useful in cursors, where you rarely use the entire image area.
- The *inverted* area reverses the color behind the icon or cursor at run time. You can use an inverted area to make part of an icon or cursor visible at all times, regardless of the color beneath it.

### Choosing a Pen Style

The pen style affects the width of a line or shape border and whether a line or shape border is solid, dashed, or dotted.

To choose a pen style:

1. In the **Tools** palette, click on the **Pen Style** tool.
2. Choose a pen style.
3. Click **OK**.

## Working with Graphics

The pen style appears in the Pen Style tool.

### Choosing a Brush Shape

You can choose a brush shape for the Paintbrush or Airbrush tool.

To choose a brush shape:

1. Select the **Paintbrush Shape** or **Airbrush Shape** tool.
2. Select a brush shape.
3. Choose **OK**.

The brush shape appears in the Paintbrush Shape or Airbrush Shape tool.

### Choosing a Paint Pattern

The paint pattern affects the interior of a filled shape and the painting styles of the Airbrush and Paintbrush tools.

To choose a paint pattern:


1. Select the **Paint Pattern** tool.
2. Select a paint pattern.
3. Choose **OK**.

The paint pattern appears in the Paint Pattern tool.

## Drawing and Painting

You can draw and paint with the Pen, Paintbrush, and Airbrush tools. The output of the Pen tool depends on the current pen style; the output of the Paintbrush and Airbrush tool depends on the current brush shape and paint pattern.

The Airbrush tool differs from the Paintbrush tool in that if you drag it slowly, the Airbrush tool paints a thick pattern, but if you drag it quickly, it paints a scattered, thinner pattern.

The size of the area covered by the Airbrush and Paintbrush tools is proportionate to the size of the drawing area. Therefore, you can zoom out to paint a larger area, and zoom in to paint a smaller area. For more information about zooming in and out,  see the section “Zooming in on or out from a Graphic” on page 680.

To draw or paint:

1. In the **Tools** palette, select the **Pen**, **Paintbrush**, or **Airbrush** tool.
2. In the drawing area, move the cursor to where you want to start drawing, and then click and hold down mouse button one (for the foreground color) or mouse button two (for the background color).

## Working with Graphics

3. Drag the cursor around the drawing area.
4. When you are done drawing or painting, release the mouse button.

### Drawing a Line

To draw a line:

1. In the **Tools** palette, select the **Line** tool.
2. In the drawing area, move the cursor to where you want the line to start, and then click and hold down mouse button one (for the foreground color) or mouse button two (for the background color).
3. To draw a line with a 0-, 45-, or 90-degree angle, hold down Shift.
4. Move the cursor to where you want the line to end, and then release the mouse button.

### Drawing a Shape

You can draw a filled or unfilled rectangle, rounded rectangle, or ellipse.

The interior of a filled shape depends on the current paint pattern, and the shape border depends on the current pen style. To draw a shape without a border, choose **Null** for the pen style.

To draw a shape:

1. In the **Tools** palette, select a shape tool.
2. In the drawing area, move the cursor to where you want a corner of the shape, and then click and hold down mouse button one (for the foreground color) or mouse button two (for the background color).
3. To draw a square or circle, hold down Shift.
4. Move the cursor to the opposite corner of the shape, and then release the mouse button.

### Filling an Area with Color

You can fill any area that is bounded by one or more different colors.

To fill an area with color:

1. In the **Tools** palette, select the **Paint Can** tool.
2. In the drawing area, move the cross hairs into the area that you want to fill, and then click mouse button one (for the foreground color) or mouse button two (for the background color).

## Working with Graphics

**Note:** Because of problems inherent in some display drivers, the Paint Can tool doesn't always work properly. To solve this problem, you can use an alternative flood-filling algorithm that is more reliable, but slower. To enable this algorithm, add the following line to the [RWS\_Icon] section of the WORKSHOP.INI file:

```
RWS_OwnFloodFill=1
```



The following example shows an edited RWS\_Icon section:

```
[RWS_Icon]
RWS_OwnFloodFill=1
PercentLeft=69
ZoomLeft=8
ZoomRight=1
bVert=1
```

## Adding Text

To add text to a graphic:

1. In the **Tools** palette, select the **Text** tool.
2. In the drawing area, click where you want to add the text.
3. Type the text.
4. To change the alignment of the text relative to the position you specified in step 2, choose **Text→Align left**, **Text→Align center**, or **Text→Align right**.
5. To change the font, choose **Text→Font**. Then specify a font name, size, and style, and then choose **OK**.
6. To change the color of the text, click on a new color in the **Colors** palette.

When you perform another action, the text becomes part of the graphic; at that point, you won't be able to select or edit the text directly. However, you can erase the area that contains the text and then add new text.


## Erasing an Area

You can erase an area by drawing or painting with the background color or by using the Eraser tool. The Eraser tool allows you to paint with the background color when you press mouse button one, and with the foreground color when you press mouse button two.

You can also fill the entire drawing area with the background color by double-clicking on the **Eraser** tool.

The size of the Eraser tool is proportionate to the size of the drawing area. Therefore, you can zoom out to erase a larger area, and zoom in to erase a smaller area. For

## Working with Graphics

more information about zooming in and out,  see the section “Zooming in on or out from a Graphic” on page 680.

### Selecting an Area

You can align, copy, move, or remove any area that you select.

To select a rectangular area:

1. In the **Tools** palette, select the **Pick Rectangle** tool.
2. In the drawing area, move the cursor to a corner of the area you want to select, and then click and hold down the mouse button.
3. Move the cursor to the opposite corner of the area you want to select and then release the mouse button.

To select a non-rectangular area:

1. In the **Tools** palette, select the **Scissors** tool.
2. In the drawing area, move the cursor to a boundary of the area you want to select, and then click and hold down the mouse button.
3. Draw a closed shape around the area you want to select, and then release the mouse button.

### Aligning an Area

You can align an area with the top, bottom, sides, or center of a graphic.

To align an area:

1. Select the area.
2. Choose **Options**→**Align**.
3. Choose a horizontal alignment option.
4. Choose a vertical alignment option.
5. Click **OK**.

**Note:** Any area left open is filled with the current background color.

### Moving or Resizing an Area

To move an area:

1. Select the area.
2. Drag the area to a new location.

To move or resize an area:

1. Select the area.
2. Choose **Options**→**Size**.

## Working with Graphics

3. To move the area, specify new pixel coordinates for the top and left sides.
4. To resize the area, specify a new height and width.
5. Click **OK**.

**Note:** Any area left open is filled with the current background color.

## Copying or Removing an Area

To copy or remove an area, select the area, and then use the **Edit→Copy**, **Edit→Paste**, and **Edit→Duplicate** commands. (The **Edit→Duplicate** command is equivalent to choosing **Edit→Copy** and **Edit→Paste**.)

**Note:** Any area left open is filled with the current background color.

## Zooming in on or out from a Graphic

To zoom in on a graphic:

- In the **Tools** palette, double-click the **Zoom** tool. Or press Ctrl+Z. Or choose **View→Zoom in**.

To zoom in on a particular area of a graphic:

1. Select the **Zoom** tool.
2. In the drawing area, move the cursor to a corner of the area, and then click and hold down the mouse button.
3. Move the cursor to the opposite corner of the area and then release the mouse button.

To zoom out from a graphic:

- In the **Tools** palette, hold down Shift and then double-click on the **Zoom** tool. Or press Ctrl+O. Or choose **View→Zoom out**.

To view the actual size of a graphic:

- Press Ctrl+A. Or choose **View→Actual size**.

## Moving a Graphic Around in the Drawing Area

If a graphic is too large for the drawing area, you can move the graphic around to see its other parts.

To move a graphic around in the drawing area:

1. Hold down Ctrl.  
The cursor becomes the hand tool.
2. Drag the graphic to a new position.

## Working with Graphics

You can also move a graphic around with the scroll bars on the right and bottom sides of the drawing area.

**Note:** You can't use the Hand tool and the Text tool at the same time.

### Viewing an Icon or Cursor in Another Resolution

To simulate the appearance of an icon or cursor in another screen resolution, choose the appropriate command from the **View** menu.

*Figure 180. Simulating the Appearance of an Icon or Cursor in Another Screen Resolution*

If you are creating this type of icon or cursor	You can view the icon or cursor in this resolution	By choosing this command
A 16x16 or 16x32 icon or a cursor with 1 bit per pixel	EGA/VGA	<b>View→EGA/VGA Resolution</b>
A 32x32 icon or a cursor with 4 bits per pixel	CGA	<b>View→CGA Resolution</b>
A 64x64 icon or a cursor with 8 bits per pixel	CGA or VGA	<b>View→CGA Resolution</b> or <b>View→EGA/VGA Resolution</b>

To view the actual size of an icon or cursor, choose **View→Actual size**.

### Testing an Icon or Cursor

To test an icon or cursor, choose the **Icon→Test** or **Cursor→Test** command. When you test an icon, the edit window for the icon is minimized and represented by the test icon. When you test a cursor, the Resource Workshop cursor becomes the test cursor.

To stop testing an icon, restore its edit window by double-clicking on the icon.

To stop testing a cursor, click anywhere.



### Adding or Removing an Image from an Icon or Cursor

In general, you create a new icon resource for each icon design (called simply an *icon*), and you don't put different icons in the same icon resource. However, it is likely you will want to put different color formats of the same icon in one icon resource. These color variations on the same icon are called *images*. For example, if you want a 2-color and a 16-color version of the same design, you can store both versions in the same icon resource.

The reason the icon resource supports different color formats is that Windows picks a color format based on the ability of the display hardware to support the format. Windows picks a 2-color format for a monochrome display driver and a 16-color format for a standard Windows VGA driver.

## Working with Graphics

To add a new image to an existing icon or cursor:

1. In the **Resource Project** window, double-click the icon or cursor.
2. Choose **Images**→**New image**.
3. If you are creating a new icon image, specify the size and maximum number of colors. If you don't know which size and color options to specify, accept the default values. For information about changing the size and color options,  see “Changing the Properties of an Icon” on page 683.
4. If you are creating a new cursor image, specify the bits per pixel. If you don't know which option to specify, accept the default values. For information about changing the bits per pixel option,  see “Changing the Properties of a Cursor” on page 683.
5. Choose **OK**.

To remove an image from an icon or cursor:

1. In the **Image** window for the icon or cursor, select the image that you want to remove.
2. Press Del.

## Changing the Properties of a Bitmap

You can change the following properties of a bitmap:

- The height and width
- Whether the bitmap is stored in a compressed format
- The maximum number of colors for the bitmap
- Whether the bitmap is stored in the Windows or OS/2 format

To change the properties of a bitmap:

1. Choose **Bitmap**→**Size and attributes**.
2. To change the size of the bitmap, type the new height and width in the appropriate boxes.
3. If you changed the height or width, but you don't want to stretch or compress the current bitmap into the new area, deselect the **Stretch current bitmap** box.
4. To save a 16- or 256-color bitmap in a compressed format, choose **RLE 4** (for 16-color bitmaps) or **RLE 8** (for 256-color bitmaps).

**Note:** Some display drivers do not support RLE compression. Also, the RLE compression algorithm can *increase* the size of bitmaps that don't have many adjacent pixels with the same color. For example, RLE compression might increase the size of a scanned photograph. Therefore, you should use RLE compression with caution.



## Working with Graphics

5. In the **Colors** group, choose the maximum number of colors for the bitmap.  
Depending on the capabilities of your display device and driver, you might be limited to two or sixteen colors.
6. In the **Format** group, choose an operating system format for the bitmap. You can use the Windows or OS/2 formats.
7. Choose **OK**.

### Changing the Properties of a Cursor

You can set the hot spot of a cursor. The hot spot is a single pixel from which the x- and y-coordinates of the cursor are determined. The location of the hot spot should be evident from the cursor image; for example, an arrow-shaped cursor should have its hot spot at the tip of the arrow.

Before you add a hot spot to a cursor, you should determine the exact coordinates for the hot spot. To pinpoint the coordinates, you can zoom in on the cursor, place a drawing tool over the pixel you want, and then note the coordinates in the status bar.

To set the hot spot:

1. Choose **Cursor→Set hot spot**.
2. Type the x- and y-coordinates for the hot spot.
3. Choose **OK**.

### Changing the Properties of an Icon

You can change the size of and the maximum number of colors for an icon:

1. Choose **Icon→Size and attributes**.
2. In the **Size** box, choose the size of the icon in pixels.
3. In the **Colors** box, choose the maximum number of colors for the icon.  
Depending on the capabilities of your display device and driver, you may be limited to two or sixteen colors.
4. Choose **OK**.

### Changing the Properties of a Font

You can change the following properties of a font:

- General properties, such as copyright and version information
- The height and width of the characters
- The number of characters
- The ANSI codes for the characters

## Working with Graphics

### Changing the General Properties of a Font

To change the general properties of a font:

1. Choose **Font→Header**.
2. Specify any of the following options:

Option	Description
<b>Face name</b>	Type a name that you want to assign to your font
<b>Device</b>	Type a device name for your font if you want to inform your programs that this font can be used only on a particular device
<b>Copyright</b>	Type copyright information for your custom font
<b>Font version</b>	Font version 2.00 is supported in all cases. You can use version 3.00 if you are creating a Windows 3.x application that will run in a protected mode environment (Standard mode or 386 Enhanced mode) on an 80386 (or later) processor.
<b>Italic</b>	The font contains italicized characters
<b>Underline</b>	The font contains underlined characters
<b>Strikeout</b>	The font contains characters that are struck out
<b>Proportional</b>	The font is a variable-width font
<b>Weight</b>	The font is of normal weight (400) or boldfaced (700)
<b>Family</b>	Describes the font family. The acceptable values are: Don't care (0), Roman (1), Swiss (2), Modern (3), Script (4), and Decorative (5).
<b>Char set</b>	Defines the character set. The value can be 0 through 255. 0, 2, and 255 have the following predefined meanings: ANSI, the default Windows character set(0); Symbol, used for math and scientific formulas (2); OEM, a machine-specific character set (255).
<b>Horiz res</b>	Horizontal number of pixels per logical inch on your video display
<b>Vert res</b>	Vertical number of pixels per logical inch on your video display
<b>Points</b>	Type size. A point is 1/72 of an inch. A character is measured from the top of the ascender to the bottom of the descender. The value you enter here should not include space for diacritical marks.

## Working with Graphics

<b>Int leading</b>	The space in pixels reserved for diacritical marks
<b>Ext leading</b>	The additional space in pixels between lines of characters
<b>Ascent</b>	The height in pixels of the character above the baseline

### Changing the Height and Width of the Characters in a Font

To change the height and width of characters in a font:

1. Choose **Font→Font size**.
2. In the **Width** box, type the width of the font. Type 0 if the width of the characters in the font can vary. (Variable-width fonts usually take less space and are more pleasing to the eye than fixed-width fonts.)
3. In the **Height** box, type the height of the font.
4. If you typed 0 for the width, type the maximum width of the characters in the font.
5. If you changed the height or width of the font, and you want to stretch or compress the existing characters into the new area, select the **Stretch current characters** box.
6. Choose **OK**.

If you use the **Font→Font size** command to specify that the width of characters in a font can vary, you can change the width of each character in the font.

To change the width of a single character in a font:

1. Select the character in the right border of the edit window.
2. Choose **Font→Font character width**.
3. Type a new width for the character.
4. If you want to stretch or compress the existing character into the new area, select the **Stretch current characters** box.
5. Choose **OK**.

### Changing the ANSI Codes of the Characters in a Font

The ANSI values to which you map a set of bitmapped images are arbitrary, but must be in the range of 0 to 255. These ANSI values become important when you load the font in your program and display the bitmaps; you use the ANSI value that corresponds to a bitmap to display it, the same as you would use an ANSI value to display a character.

To change the ANSI codes of the characters in a font:

1. Choose **Font→Font size**.
2. In the **First** box, type the ANSI code for the first character in the font.


## Working with Graphics

3. In the **Last** box, type the ANSI code for the last character in the font.
4. In the **Default** box, type the ANSI code for the character that appears first when you load the font into the Graphic editor.
5. In the **Break** box, type the ANSI code for the break character. (The break character, typically a space, is used to pad justified lines.)
6. Choose **OK**.

**Changing the Number of Characters in a Font** To change the number of characters in a font, change the ANSI code for the first or last character in the font. (This procedure is described in the previous section.) The number of characters in the font equals the difference between the ANSI codes of the first and last characters.

### Editing a Specific Character in a Font

When you load a font into the graphic editor, the first character of the font appears automatically in the **Edit** window. To bring another character into the **Edit** window, click the character in the right border of the **Edit** window.

For more information about fonts,  see the section “Changing the Properties of a Font” on page 683.

---

## Customizing the Graphic editor

You can customize the Graphic editor by

- Adding or removing a second pane from the **Edit** window
- Hiding or showing the **Colors** and **Tools** palettes
- Changing a color on the **Colors** palette
- Specifying whether the **Colors** palette is saved when you exit Resource Workshop
- Specifying whether a grid appears in the drawing area when you zoom in on a graphic

### Adding or Removing a Second Pane from the Edit Window

In the Graphic editor, you can look at two different views of the image you are creating or editing. You can split the window vertically or horizontally to show the two views side-by-side or one view above the other. You can also choose how to zoom in on or out from each view.

To split the window, choose **View→Split horizontal** or **View→Split vertical**.

When the window is split, one of the panes is *active*. The active pane is the one in which you are working. To make a pane active, click on it.

## Working with Graphics

To see more of one pane than the other, move the cursor to the line that splits the images (the separator bar) and, when the cursor becomes a double arrow, drag the separator bar.

To use a single-pane view, drag the separator bar to the far left or right (for a vertical split) or to the top or bottom (for a horizontal split).

If you add a second pane to the **Edit** window, you can specify whether changes are reflected in the inactive pane *while* you draw or paint or when you release the mouse button *after* drawing or painting.

To specify when changes are reflected in the inactive pane:

1. Choose **Options**→**Editor options**.
2. Select or deselect the **Draw on both images** box.
3. Choose **OK**.

### Hiding or Showing the Tools and Colors Palettes

To hide the **Tools** or **Colors** palette, click on **Hide** in the **System** menu in the upper left corner of the palette.

You can also hide the **Tools** or **Colors** palette by choosing the **Hide toolbox** or **Hide palette** command. The name of the menu where you will find this command depends on the type of resource you are currently editing. For example, if you are editing an icon, the **Hide palette** command is in the **Icon** menu. If you are editing a cursor, the menu is in the same position on the menu bar, but it is called the **Cursor** menu.

To show the **Tools** or **Colors** palette, choose the **Show palette** or **Show toolbox** command on the appropriate menu.

### Changing a Color on the Colors Palette

To change a color on the **Colors** palette:

1. In the **Colors** palette, double-click on the color.
2. In the **Red**, **Green**, and **Blue** boxes, type values from 0 to 255 to indicate the intensities of each color. (The colors are mixed together to form the new color.)
3. To return the color to its default value, choose **Default** (if your display device and driver don't support 256 colors) or **System** (if your display device and driver do support 256 colors.)
4. Choose **OK**.

**Note:** You can't change the **Colors** palette when you are editing a font or cursor.

## Working with Graphics

### Specifying Whether the Colors Palette is Saved

To specify whether the **Colors** palette is saved when you exit Resource Workshop:

1. Choose **Options**→**Editor options**.
2. Select or deselect the **Save with default colors** box. (If you deselect the box, any changes you make to the **Colors** palette will be saved when you exit Resource Workshop.)
3. Choose **OK**.

### Specifying Whether a Grid Appears When You Zoom in

To specify whether a grid appears in the drawing area when you zoom in on a graphic:

1. Choose **Options**→**Editor options**.
2. Select or unselect the **Grid on zoomed windows** box.
3. Choose **OK**.

**Note:** The grid may not appear if you are using 256 colors.

---

## Using a Graphic Resource in an Application

You can include graphics in an application using the Windows API.

### Programming a Bitmap with the Windows API



To program a bitmap with the Windows API, use the `LoadBitmap`, `GetObject`, `GetDC`, `CreateCompatibleDC`, `BitBlt`, `DeleteDC`, and `ReleaseDC` functions. For example, the following code displays a bitmap in the window represented by the *HWnd* variable.

## Working with Graphics

```
HINSTANCE hInst;
HWND hWnd;
HDC hDC, hDCMemory;
HBITMAP hBitmap, hTempBitmap;
BITMAP bitmap;
// Get the bitmap resource and its handle.
hBitmap = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP1));
// Get the bitmap data.
GetObject(hBitmap, sizeof(BITMAP), &bitmap);
// Get the device context of the target window.
hDC = GetDC(hWnd);
// Create a compatible memory device context
hDCMemory = CreateCompatibleDC(hDC);
// Copy the bitmap into the memory device context.
hTempBitmap = SelectObject(hDCMemory, hBitmap);
// Copy the bitmap into the device context of the target window.
BitBlt(hDC, 0, 0, bitmap.bmWidth, bitmap.bmHeight, hDCMemory, 0, 0, SRCCOPY);
// Clean up and exit.
SelectObject(hDCMemory, hTempBitmap);
DeleteDC(hDCMemory);
ReleaseDC(hWnd, hDC);
```

In the preceding code, IDB\_BITMAP1 is the identifier for the bitmap.

### Programming an Icon with the Windows API



To program an icon with the Windows API, use the LoadIcon function. For example, the following code loads an icon resource into a variable:

```
HICON hIcon;
hIcon = LoadIcon(hInst, MAKEINTRESOURCE(IDD_ICON1))
```

In the preceding code, IDD\_ICON1 is the identifier for the icon.

### Programming a Cursor with the Windows API



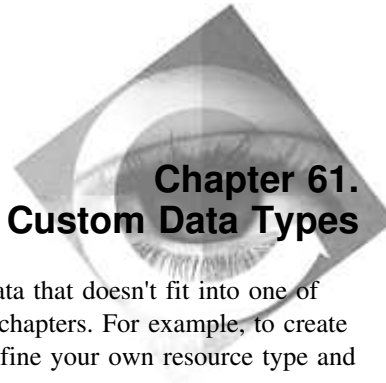
To program a cursor with the Windows API, use the LoadCursor and SetCursor functions. For example, the following code changes the active cursor:

```
HCURSOR hCursor;
hCursor = LoadCursor(hInst, MAKEINTRESOURCE(IDD_CURSOR1))
SetCursor(hCursor)
```

In the preceding code, IDD\_CURSOR1 is the identifier for the cursor.

## **Working with Graphics**





## Chapter 61. Version Information and Custom Data Types

You can define your own resource types to contain data that doesn't fit into one of the standard resource types described in the previous chapters. For example, to create a string that is longer than 255 characters, you can define your own resource type and store your long strings there.

You can also include *metafiles* in your project as user-defined resources. A metafile is a graphic in source form. It is made up of a collection of Graphics Device Interface (GDI) calls. Metafiles are easier to scale and more device-independent than standard bitmaps, and often require less memory than a bitmap resource.

Like other resources, you define version information or a custom data type in a resource script. This chapter describes how to edit resource scripts for version information and custom data types, and how to use the corresponding resources in an application.

---

### Creating a Custom Data Type

To create a custom data type:

1. Choose **Resource**→**New**.
2. Choose **New Type**.
3. Type the name of the new data type.
4. Choose **OK**.
5. Choose **Yes** to create a new identifier.
6. Type a value for the identifier.
7. In the **File** box, choose a header file in which to store the identifier.
8. Choose **OK**.

---

### Editing a Resource Script for Version Information or a Custom Data Type

To edit a resource script for version information or a custom data type, you must start the script editor.

To start the script editor:

## Version information and custom data types

1. Add a new resource script for version information or a custom data type. Or open a resource script file that contains a resource script for version information or a custom data type.
2. If the script editor doesn't start automatically, double-click on the version information or custom data type in the **Resource Project** window.

The script editor appears.

**Note:** You can also use an external text editor to edit a resource script for version information or a custom data type.

---

## Editing a Resource Script for Version Information

A resource script for version information has the following syntax:

```
versionID VERSIONINFO fixed-info
{
    block-statements
}
```

**The versionID Parameter** The *version ID* parameter is the resource identifier. It must be set to 1.

**The fixed-info Parameter** The *fixed-info* parameter specifies version information about the file to which the version information resource is linked. It consists of the following statements:

Parameter	Description
<b>FILEVERSION</b> <i>version</i>	A file version number, which is a 64-bit integer. You can define the 64-bit integer with four 16-bit integers separated by commas. For example, "10,11,0,1" represents 0x000a 000b 0000 0001.
<b>PRODUCTVERSION</b> <i>version</i>	A product version number, which is a 64-bit integer. You can define the 64-bit integer with four 16-bit integers separated by commas. For example, "10,11,0,1" represents 0x000a 000b 0000 0001.
<b>FILEFLAGSMASK</b> <i>fileflags</i>	The valid bits in FILEFLAGS statement. Use any of the <i>fileflags</i> constants, which are described in the section "Constants" on page 694.
<b>FILEFLAGS</b> <i>fileflags</i>	The Boolean attributes of a file. Use any of the <i>fileflags</i> constants, which are described in the section "Constants" on page 694.

## Version information and custom data types

<b>FILEOS</b> <i>fileos</i>	The operating system for a file. Use any of the <i>fileos</i> constants, which are described in the section “Constants” on page 694. The values 0x00002L, 0x00003L, 0x20000L, and 0x3000L are reserved.
<b>FILETYPE</b> <i>filetype</i>	A file type. Use any of the <i>filetype</i> constants, which are described in the section “Constants” on page 694. All other values are reserved.
<b>FILESUBTYPE</b> <i>subtype</i>	The function of a file. The <i>subtype</i> parameter is 0 unless the FILETYPE statement is FILETYPE VFT_DRV, VFT_FONT, or VFT_VXD. Use any of the <i>subtype</i> constants, which are described in the section “Constants” on page 694.

### The block-statements Parameter

The *block-statements* parameter specifies string or variable information about the file to which the version information resource is linked.

### A String Information Block

A string information block has the following syntax:

```
BLOCK "StringFileInfo"
{
    BLOCK "lang-charset" VALUE "string-name", "value"
    .
    .
    .
}
```

A string information block has the following parameters:

Parameter	Description
<i>lang-charset</i>	A string containing a hexadecimal number that represents a character set. The string is a concatenation of the <i>langID</i> and <i>charsetID</i> constants, which are described in the section “Constants” on page 694.
<i>string-name</i>	The name of the corresponding value in the VALUE statement. Use a custom string or one of the string-name constants, which are described in the section “Constants” on page 694.
<i>value</i>	The value of the corresponding string-name in the VALUE statement

### A Variable Information Block

A variable information block has the following syntax:

## Version information and custom data types

```
BLOCK "VarFileInfo"
{
    VALUE "Translation",
        langID, charsetID
    .
    .
    .
}
```

A variable information block has the following parameters:

Parameter	Description
<i>langID</i>	A string containing a hexadecimal number that represents a language. Use any of the <i>langID</i> constants, which are described in the section “Constants.”
<i>charsetID</i>	A string containing a hexadecimal number that represents a character set. Use any of the <i>charsetID</i> constants, which are described in the section “Constants.”

**Constants** If you include the `ver.h` file in the resource script file that contains the version information resource, you can use constants for the *fileflags*, *fileos*, *filetype*, *subtype*, *langID*, and *charsetID* parameters.

**fileflags:** You can specify the following constants for the *fileflags* parameter:

Constant	Description
<b>VS_FF_DEBUG</b>	Debugging--file contains debugging information or was compiled with debugging features enabled
<b>VS_FF_INFOINFERRED</b>	Incorrect version information--file contains a dynamically created version information resource with possible empty or incorrect blocks. Do not use this value in version information resources created with the <code>VERSIONINFO</code> statement.
<b>VS_FF_PATCHED</b>	Patch--file has been modified; it is not identical to the original shipping file of the same version number
<b>VS_FF_PRERELEASE</b>	Prerelease--file is a development version, not a commercially released product
<b>VS_FF_PRIVATEBUILD</b>	Private build--file was not built using standard release procedures. When you use this value, include the <code>PrivateBuild</code> string in the <code>string- name</code> parameter.

## Version information and custom data types

**VS\_FF\_SPECIALBUILD** Special build--file was built by the original company using standard release procedures, but is a variation of the standard file of the same version number. When you use this value, include the SpecialBuild string in the string-name parameter.

**fileos:** You can specify the following constants for the *fileos* parameter:

Constant	Description
<b>VOS_UNKNOWN</b>	File designed for unknown operating system
<b>VOS_DOS</b>	File designed for MS-DOS
<b>VOS_NT</b>	File designed for Windows NT
<b>VOS_WINDOWS16</b>	File designed for Windows 3.0 or later
<b>VOS_WINDOWS32</b>	File designed for Windows 32-bit
<b>VOS_DOS_WINDOWS16</b>	File designed for Windows 3.0 or later running on MS-DOS
<b>VOS_DOS_WINDOWS32</b>	File designed for Windows 32-bit running on MS-DOS
<b>VOS_NT_WINDOWS32</b>	File designed for Windows 32-bit running on Windows NT

**filetype:** You can specify any of the following constants for the *filetype* parameter:

Constant	Description
<b>VFT_UNKNOWN</b>	File type is unknown to Windows
<b>VFT_APP</b>	File contains an application
<b>VFT_DLL</b>	File contains a dynamic-link library
<b>VFT_DRV</b>	File contains a device driver. If you use this constant, include a more specific description of the driver in FILESUBTYPE.
<b>VFT_FONT</b>	File contains a font. If you use this constant, include a more specific description of the font file in FILESUBTYPE.
<b>VFT_VXD</b>	File contains a virtual device. If you use this constant, include a more specific description of the device in FILESUBTYPE.
<b>VFT_STATIC_LIB</b>	File contains a static-link library

## Version information and custom data types

**subtype:** If the FILETYPE statement specifies VFT\_DRV, you can specify any of the following constants for the *subtype* parameter:

Constant	Description
VFT2_UNKNOWN	Driver type is unknown to Windows
VFT2_DRV_COMM	File contains a communications driver
VFT2_DRV_PRINTER	File contains a printer driver
VFT2_DRV_KEYBOARD	File contains a keyboard driver
VFT2_DRV_LANGUAGE	File contains a language driver
VFT2_DRV_DISPLAY	File contains a display driver
VFT2_DRV_MOUSE	File contains a mouse driver
VFT2_DRV_NETWORK	File contains a network driver
VFT2_DRV_SYSTEM	File contains a system driver
VFT2_DRV_INSTALLABLE	File contains an installable driver
VFT2_DRV_SOUND	File contains a sound driver

If the FILETYPE statement specifies VFT\_FONT, you can specify any of the following constants for the *subtype* parameter:

Constant	Description
VFT2_UNKNOWN	Font type is unknown to Windows
VFT2_FONT_RASTER	File contains a raster font
VFT2_FONT_VECTOR	File contains a vector font
VFT2_FONT_TRUETYPE	File contains a TrueType font

If the FILETYPE statement specifies VFT\_VXD, you can't specify a value for the subtype statement.

**langID:** You can specify any of the following constants for the *langID* parameter:

Figure 181 (Page 1 of 2). *langID* parameter

Constant	Description	Constant	Description
0x0401	Arabic	0x0418	Romanian
0x0402	Bulgarian	0x0419	Russian
0x0403	Catalan	0x041A	Croato-Serbian (Latin)
0x0404	Traditional Chinese	0x041B	Slovak

## Version information and custom data types

Figure 181 (Page 2 of 2). langID parameter

Constant	Description	Constant	Description
0x0405	Czech	0x041C	Albanian
0x0406	Danish	0x041D	Swedish
0x0407	German	0x041E	Thai
0x0408	Greek	0x041F	Turkish
0x0409	U.S. English	0x0420	Urdu
0x040A	Castilian Spanish	0x0421	Bahasa
0x040B	Finnish	0x0804	Simplified Chinese
0x040C	French	0x0807	Swiss German
0x040D	Hebrew	0x0809	U.K. English
0x040E	Hungarian	0x080A	Mexican Spanish
0x040F	Icelandic	0x080C	Belgian French
0x0410	Italian	0x0810	Swiss Italian
0x0411	Japanese	0x0813	Belgian Dutch
0x0412	Korean	0x0814	Norwegian-Nynorsk
0x0413	Dutch	0x0816	Portuguese
0x0414	Norwegian-Bokmal	0x081A	Serbo-Croatian (Cyrillic)
0x0415	Polish	0x0c0C	Canadian French
0x0416	Brazilian Portuguese	0x100C	Swiss French
0x0417	Rhaeto-Romanic		

**charsetID:** You can specify any of the following constants for the *charsetID* parameter:

Constant	Description
<b>0</b>	7-bit ASCII
<b>932</b>	Windows, Japan (Shift - JIS X-0208)
<b>949</b>	Windows, Korea (Shift - KSC 5601)
<b>950</b>	Windows, Taiwan (GB5)
<b>1200</b>	Unicode
<b>1250</b>	Windows, Latin-2 (Eastern European)
<b>1251</b>	Windows, Cyrillic
<b>1252</b>	Windows, Multilingual

## Version information and custom data types

<b>1253</b>	Windows, Greek
<b>1254</b>	Windows, Turkish
<b>1255</b>	Windows, Hebrew
<b>1256</b>	Windows, Arabic

***string-name*:** You can specify any of the following constants for the *string-name* parameter:

<b>Constant</b>	<b>Description</b>
<b>Comments</b>	Additional information for diagnostic purposes. Optional.
<b>CompanyName</b>	The company that produced the file. Required.
<b>FileDescription</b>	File description. You can display this string in a list box during installation. Required.
<b>FileVersion</b>	File version number. Required.
<b>InternalName</b>	File internal name. If file does not have internal name, use original filename, without extension. Required.
<b>LegalCopyright</b>	Legal copyright notices. Optional.
<b>LegalTrademarks</b>	Trademarks and registered trademarks that apply to file. Optional.
<b>OriginalFilename</b>	Original file name, not including path. Required.
<b>PrivateBuild</b>	Information about private version of file. Required if VS_FF_PRIVATEBUILD is set in FILEFLAGS.
<b>ProductName</b>	Name of product that file is distributed with. Required.
<b>ProductVersion</b>	Version of product that file is distributed with. Required.
<b>Special Build</b>	Specifies how this version of file differs from standard version. Required if VS_FF_SPECIALBUILD is set in FILEFLAGS.

## Editing a Resource Script for a Custom Data Type

A resource script for a custom data type has the following syntax:

```
resource-name type-ID [load-type] [memory-option] filename
{
    block-statements
}
```



## Version information and custom data types

### The resource-name Parameter

The *resource-name* identifier for the resource.

### The type-ID Parameter

The *type-ID* parameter sets the resource type. You must set the *type-ID* to a number greater than 255. (Numbers 1 to 255 are reserved by Windows for predefined resource types and future expansion.)

### The load-type Parameter


The way in which the resource is loaded into memory. You can set the *load-type* parameter to:

- PRELOAD, which means the resource is loaded when its application starts
- LOADONCALL, which means the resource is loaded when it is needed

### The filename Parameter

The name of the DOS file containing the user data. A full path name can be used to specify files which are not in the current working directory. The data in the specified file is included in the current project.

### The block-statements Parameter

You can specify string or variable blocks. For more information,  see “Editing a Resource Script for Version Information or a Custom Data Type” on page 691.

## Entering Data in a Resource Script

Here are some guidelines for specifying data between the begin ({) and end (}) parameter:

- The data can include any combination of numeric values and strings.
- You can use hexadecimal, octal, or decimal notation to represent numeric values and strings.
  - Use 0x (a zero followed by the letter *x*) or \$ (a dollar sign) as the leading characters for hexadecimal notation. This notation supports only 16-bit values. If you want to use an odd number of hexadecimal values, use the *hexstring* data type (described in the next bullet).
  - Use 0o (a zero followed by the letter *o*) or just 0 (zero) as the leading characters for octal notation.
- You can also represent hexadecimal values by using a *hexstring* of hexadecimal values enclosed in single quotation marks. The compiler ignores any spaces you insert to make the hex codes more readable. For example, you could represent the ASCII values of the characters in the word *string* as '73 74 72 69 6E 67' and the hex number 06E07A as '06E07A'.
- If you include text strings in your resource, enclose the strings in quotation marks, like this: "string". Strings aren't automatically null terminated. To

## Version information and custom data types

terminate a string with a null character, type `\0` (backslash zero) at the end of the string.

For example, here is what a resource script might look like:

```
RESTYPE_1 RESTYPE
{
    "This is a string."
    0xFFAA 0o7076 01077
    '54 68 69 73 20 69 73 0D 0A 73 6F 6D 65 20 73 61'
    '6D 70 6C 65 0D 64 61 74 61 2E'
}
```

---

## Compiling a Resource Script

You can compile a resource script by choosing **Compile**→**Compile now** or by pressing **Alt+F9**.

If there is a compilation error, Resource Workshop displays an error message. Read the error message and click on **OK** when you are done. Resource Workshop puts you back into the script editor so that you can fix the error.

---

## Using a Version Information Resource in an Application

To include a version information resource in an application, use any of the version functions.



For example, the following code loads the version information data into variables:

```
LPBYTE companyName;
char appFileName[255];
BYTE versionInfoData[1024];
DWORD versionInfoHandle;
UINT dwSize;
UINT vSize;
LPBYTE translationBlock;
char blockName[255]
GetModuleFileName(hInst, appFileName, sizeof(appFileName));
dwSize = (UINT) ::GetFileVersionInfoSize(appFileName, &versionInfoHandle);
if (dwSize)
{
    ::GetFileVersionInfo(appFileName, versionInfoHandle, dwSize, versionInfoData);
    strcpy(blockName, "\\VarFileInfo\\Translation");
    ::VerQueryValue(versionInfoData, blockName, (void FAR* FAR*) &translationBlock,
        &vSize);
    *(DWORD *)translationBlock = MAKELONG(HIWORD(*(DWORD *)translationBlock),
        LOWORD(*(DWORD *)translationBlock));
    wsprintf(blockName, "\\StringFileInfo\\%08lx\\CompanyName", *(DWORD
        *)translationBlock);
    ::VerQueryValue(versionInfoData, blockName (void FAR* FAR*) &companyName, &vSize);
}
```

## Version information and custom data types

### Using a Resource for a Custom Data Type in an Application



To include a custom data type resource in an application, use the `FindResource`, `LockResource`, and `UnlockResource` functions. For example, the following code loads and unloads a resource for a custom data type.

```
HRSRC hRCData;
HGLOBAL hRCDataMem;
BYTE * pRCDataMem;
// Get a handle for the resource.
hRCData = FindResource(hInst, MAKEINTRESOURCE(IDR_CUSTOM1));
// Load the resource into global memory.
hRCDataMem = LoadResource(hInst, hRCData);
// Lock the resource into memory so it can be used.
pRCDataMem = (BYTE*)LockResource(hRCDataMem);
// Replace this line with code that uses the resource.
// Unlock and free the memory used by the resource.
UnlockResource(hRCDataMem);
FreeResource(hRCDataMem);
```

## **Version information and custom data types**



## Chapter 62. Help Table and Help Subtable Resources

The help table resource contains an entry for each application window for which IPF help is provided. The help table associates each application window with its corresponding help subtable and the window identifier of its extended help window. The HELPTABLE statement defines the contents of a help table resource.

The help subtable resource contains an entry for each item that can be selected in an application window. The help subtable associates each entry field, menu item, and push button within the window with the window identifier (ID) of its help window. When the user requests help on a field, menu item, or push button in the application window, IPF uses the help subtable associated with the field to find the window ID of the contextual help window for the field. The HELPSUBTABLE statement defines the contents of a help subtable resource.

Like other resources, you define a help table or help subtable with a resource script. This chapter shows how to create and edit a resource script for a help table or help subtable resource, and explains the syntax of the HELPTABLE and HELPSUBTABLE statements.

---

### HELPTABLE Statement

You can provide any number of HELPTABLE statements in a resource script file, but each statement must specify a unique helptable-id value. You can also provide any number of helpitem-definition statements in the help table. These statements specify the application windows for which help is provided, the help subtable associated with each application window, and the extended help panels associated with each application panel.

### Syntax

The syntax of the HELPTABLE statement is:

```
helptable-id HELPTABLE
  BEGIN
    helpitem-definition
    .
    .
    .
  END
```

where:

## Help Table and Help Subtable Resources

<b>helptable-id</b>	Specifies the resource identifier of the help table. This value must be an integer in the range 1 through 65535, or a simple expression that evaluates to a value in that range.
<b>helpitem-definition</b>	Specifies an application window and the associated help subtable and extended help panel.

### Example



This example creates a help table resource whose helptable-id is `HELP_TABLE`. Each helpitem-definition statement specifies an application window, a help subtable, and an extended help panel.

```
HELP_TABLE HELPTABLE
BEGIN
    WND_MAIN,          SUBTABLE_MAIN,    100
    WND_TEXTDIALOG,    SUBTABLE_DIALOG,  200
END
```

---

## HELPSUBTABLE Statement

You can specify any number of HELPSUBTABLE statements in a resource script file, but each statement must specify a unique helptsubtable-id value. You can also provide any number of helptsubitem-definition statements in the help subtable. These specify the child window for which help is provided and the help-panel id.

### Syntax

The syntax of the HELPSUBTABLE statement is:

```
helptsubtable-id HELPSUBTABLE
BEGIN
    helptsubitem-definition
    .
    .
    .
END
```

where:

<b>helptsubtable-id</b>	Specifies the resource identifier of the help subtable. This value must be an integer in the range 1 through 65535, or a simple expression that evaluates to a value in that range.
<b>helptsubitem-definition</b>	Specifies a child window for which help is provided and the help-panel id.

## Help Table and Help Subtable Resources

### Example



This example creates two help subtable resources whose identifiers are SUBTABLE\_MAIN and SUBTABLE\_DIALOG. Each helpsubitem-definition statement specifies a child window and a help panel.

```
SUBTABLE_MAIN HELPSUBTABLE
BEGIN
    WND_HELLO, 100
    WND_LISTBOX, 102
    MI_EDIT, 110
    MI_ALIGNMENT, 111
    MI_LEFT, 112
    MI_CENTER, 113
    MI_RIGHT, 114
    MI_TEXT, 199
END

SUBTABLE_DIALOG HELPSUBTABLE
BEGIN
    DID_ENTRY, 201
    DID_OK, 202
    DID_CANCEL, 203
END
```

---

### Creating a Help Table Resource

To create a help table resource:

1. Choose **Resource**→**New**.
2. Choose **New Type**.
3. In the **Resource type** field, enter **HELPTABLE**.
4. Choose **OK**.
5. Choose **Yes** to create a new identifier.
6. In the **Value** field, enter **HELPTABLE**.
7. In the **File** box, choose a header file in which to store the identifier.
8. Choose **OK**.

---

### Creating a Help Subtable Resource

To create a help subtable resource:

1. Choose **Resource**→**New**.
2. Choose **New Type**.

## Help Table and Help Subtable Resources

3. In the **Resource type** field, enter **HELPSUBTABLE**.
4. Choose **OK**.
5. Choose **Yes** to create a new identifier.
6. In the **Value** field, enter **HELPSUBTABLE**.
7. In the **File** box, choose a header file in which to store the identifier.
8. Choose **OK**.

---

## Editing a Help Table or Help Subtable Resource

To edit a help table or help subtable resource, you must start the script editor.

To start the script editor:

1. Add a new help table or help subtable resource. Or open a resource script file that contains a resource script for a help table or help subtable.
2. If the script editor doesn't start automatically, double-click on a help table or help subtable in the **Resource Project** window.

The script editor appears

**Note:** You can also use an external text editor to edit a resource script for help table or help subtable information.



---

## Part 12. Additional Utilities You May Find Useful

VisualAge for C++ provides a number of utilities that can help you complete your applications and make programming tasks easier:

<b>NMAKE</b>	Builds your application based on dependencies and rules
<b>Message Compiler</b>	Binds messages to your application
<b>IPFC</b>	Creates online documentation.
<b>CPPFILT</b>	Demangles compiled C++ names
<b>Resource Image Conversion Utility</b>	Converts icons, pointers, cursors, and bitmaps from OS/2 format to Windows format, and vice versa

---

<b>Chapter 63. Program Maintenance Utility (NMAKE)</b>	709
Why Use NMAKE?	709
Running NMAKE	709
Options	712
Description Files	715
Macros	718
Special Macros	721
Inference Rules	725
Inference Rules Example	726
Directives	727
Inline Files	732
Characters That Modify Commands	734
Macros and Inference Rules in TOOLS.INI	737
<b>Chapter 64. Using the Message Compiler</b>	739
MC Syntax	739
Input Message File	740
Message Win32 API Calls	746
<b>Chapter 65. Creating Online Documentation</b>	751
<b>Chapter 66. Demangling Compiled C++ Names with CPPFILT</b>	753
Using the CPPFILT Utility	753
Text Mode	753
Text Mode Options	754
Binary Mode	757
Binary Mode Options	758

<b>Chapter 67. Resource Image Conversion Utility</b> . . . . .	763
Starting the Utility . . . . .	763
Using the Utility . . . . .	764

---



## Chapter 63. Program Maintenance Utility (NMAKE)

The Program Maintenance Utility (NMAKE) automates the process of updating project files. NMAKE compares the modification dates for one set of files (the target files) with those of another set of files (the dependent files). If any dependent files have changed more recently than the target files, NMAKE executes a series of commands to bring the targets up-to-date.

---

### Why Use NMAKE?

The most common use of NMAKE is to automate the process of updating a project after you make a change to a source file. Large projects tend to have many source files. Often, only a few of your source files need to be compiled when you make a change. You set up a special text file, called a *description* file (or *makefile*), that tells NMAKE:

- Which files depend on others
- Which commands, such as compile and link commands, need to be carried out to bring your program up-to-date

This use of NMAKE is only one example of its power. By building suitable description files, you can use NMAKE to

- Make backups
- Configure data files
- Run programs when data files are modified

---

### Running NMAKE

Run NMAKE by typing `nmake` on the operating-system command line. Supply input to NMAKE by either of two methods:

- Enter the input directly on the command line.
- Put your input into a *command file* (a text file, also called a *response file*) and enter the file name on the command line.

Press Ctrl+C at any time during an NMAKE run to return to the operating system.

**Note:** Do not use the ampersand character (&) to combine the NMAKE command with the `cd`, `chdir`, or `set` command.

## Program Maintenance Utility (NMAKE)

### Using the Command Line

When using NMAKE at the command line, keep the following in mind:

- All fields are optional.
- NMAKE always looks first in the current directory for a description file called `makefile`. If `makefile` does not exist, NMAKE uses the *filename* given with the `/F` (specify description file) option (see “Specify Description File (/F)” on page 713).

### Command-Line Syntax

```
nmake [options] [macrodefinitions] [targets] [/F filename]
```

where:

**options** Specifies options that modify NMAKE's actions.

**macrodefinitions**

Lists macro definitions for NMAKE to use. Macro definitions that contain spaces must be enclosed by double quotation marks.

**targets** Specifies the names of one or more target files to build. If you do not list any targets, NMAKE builds the first target in the description file.

**/F *filename*** Gives the name of the description file where you specify file dependencies and which commands to execute when a file is out-of-date.



The following example:

```
nmake /s "program = flash" sort.exe search.exe
```

- Invokes NMAKE with the `/s` option
- Defines a macro, assigning the string "flash" to the macro "program"
- Specifies two targets: `sort.exe` and `search.exe`

By default, NMAKE uses the file named `makefile` as the description file.

## Program Maintenance Utility (NMAKE)

### Command-Line Help

To display NMAKE help, type `nmake /?` at the prompt. The appropriate copyright statement appears, along with the following:

Usage:

```
NMAKE @commandfile
NMAKE /help

NMAKE [/nologo] [/acdeinpqrst?] [/f makefile] [/x stderrfile]

[macrodefs][targets]
```

What the options stand for

```
/a      Force all targets to be built
/c      Cryptic mode; suppress sign-on banner & warning messages
/d      Display modification dates
/e      Environment variables override macros in the makefile
/i      Ignore exit codes of commands invoked
/n      No execute mode; display commands only
/p      Print macro definitions & target descriptions
/q      Query if target is up to date; for use in batch files
/r      Inference Rules from 'TOOLS.INI' to be ignored
/s      Silent execution of commands
/t      Touch targets with current date & time
/?      Help message
/help   Help message
/nologo Do not display sign-on banner
```

### Using NMAKE Command Files


A command file is a *response file* used to extend command-line input to NMAKE.

You can split input to NMAKE between the command line and a command file. Use the name of a command file (preceded by `@`) where you normally type the input information on the command line.

#### Why Use a Command File?

Use a command file for

- Complex and long commands you type frequently
- Strings of command-line arguments, such as macro definitions, that exceed the limit for command-line length

**Note:** A command file is not the same as a description file. For information about description files,  see “Description Files” on page 715.

## Program Maintenance Utility (NMAKE)

### Command File Syntax

To provide input to NMAKE with a command file, type

```
nmake @commandfile
```

In the *commandfile* field, enter the name of a file containing the same information as is normally entered on the command line.

NMAKE treats line breaks that occur between arguments as spaces. Macro definitions can span multiple lines if you end each line except the last with a backslash (\). Macro definitions that contain spaces must be enclosed by quotation marks, just as if they were entered directly on the command line.

### Example



The following is a command file called update:

```
/s "program \  
= flash" sort.exe search.exe
```

You can use this command file by typing the following command:

```
nmake @update
```

This runs NMAKE using:

- The /s option
- The macro definition "program = flash"
- The targets specified as sort.exe and search.exe
- The description file makefile by default

Note that the backslash allows the macro definition to span two lines.

---

## Options

You can use several options with NMAKE. Keep the following in mind when using options:

- Option characters are not case sensitive; /I and /i are equivalent.
- You can use either a slash or dash before the option characters; -a and /a are equivalent.

### Produce Error File (/X)

**Syntax:** /X stderrfile

This option produces a standard error file.

## Program Maintenance Utility (NMAKE)

### Build All Targets (/A)

**Syntax:** /A

This option builds all specified targets even if they are not out-of-date with respect to their dependent files.

 See “Description Files” on page 715.

### Suppress Messages (/C)

**Syntax:** /C

This option suppresses display of the NMAKE sign-on banner, nonfatal error messages, and warning messages. To suppress the sign-on banner without suppressing other messages, use the /NOLOGO option.

### Display Modification Dates (/D)

**Syntax:** /D

This option displays the modification date of each file when the dates of target and dependent files are checked.

 See “Description Files” on page 715.

### Override Environment Variables (/E)

**Syntax:** /E

This option disables inherited macro redefinition.

NMAKE *inherits* all current environment variables as macros, which can be redefined in a description file. The /E option disables any redefinition — the inherited macro always has the value of the environment variable.

### Specify Description File (/F)

**Syntax:** /F *filename*

This option specifies *filename* as the name of the description file to use. If a dash (-) is entered instead of a file name, NMAKE reads a description file from the standard input device, typically the keyboard.

If a filename is not specified, it defaults to makefile.

## Program Maintenance Utility (NMAKE)

### Display Help (/HELP or /?)

**Syntax:** /HELP or /?

This option displays a brief summary of NMAKE syntax.

### Ignore Exit Codes (/I)

**Syntax:** /I

This option ignores exit codes (also called error level or return codes) returned by programs such as compilers or linkers called by NMAKE. If this option is not specified, NMAKE ends when any program returns a nonzero exit code.

### Display Commands (/N)

**Syntax:** /N

This option causes NMAKE commands to be displayed but not executed. Use the /N option to:

- Check which targets are out-of-date with respect to their dependents
- Debug description files

### Suppress Sign-On Banner (/NOLOGO)

**Syntax:** /NOLOGO

This option suppresses the sign-on banner display when NMAKE is started. If you want to suppress nonfatal error messages and warnings as well, use the suppress messages (/C) option.

### Print Macro and Target Definitions (/P)

**Syntax:** /P

This option writes out all macro definitions and target definitions. Output is sent to the standard output device (typically the display).

### Return Exit Code (/Q)

**Syntax:** /Q

This option causes NMAKE to return either of the following:

- An exit code of zero if all targets built during an NMAKE run are up-to-date
- An exit code other than zero if they are not up-to-date

Use this option to run NMAKE from within a batch file.



## Program Maintenance Utility (NMAKE)

### Ignore TOOLS.INI File (/R)

**Syntax:** /R

This option ignores the following:

- All inference rules and macros contained in the TOOLS.INI file
- All predefined inference rules and macros

### Suppress Command Display (/S)

**Syntax:** /S

This option suppresses the display of commands as they are executed by NMAKE. It does not suppress the display of messages generated by the commands themselves.

The /N command (Display Commands) takes precedence over the /S option. If you use /N and /S together, commands are displayed but not executed.

### Change Target Modification Dates (/T)

**Syntax:** /T

This option changes or “touches” the modification dates for out-of-date target files to the current date. No commands are executed, and the target file is left unchanged.

---

## Description Files

NMAKE uses a description file to determine what to do. In its simplest form, a description file tells NMAKE which files depend on others and which commands need to be executed if a file changes.

A description file looks like this:

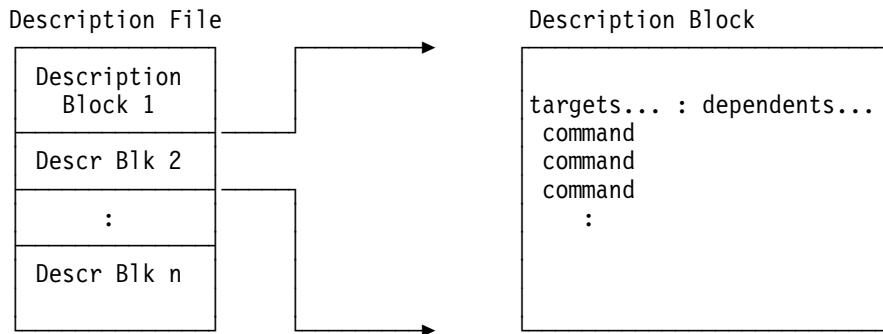
```
targets...: dependents...
      command
      :

targets... : dependents...
      command
```

## Description Blocks

A dependent relationship between files is defined in a *description block*. A description block indicates the relationship among various parts of the program. It contains commands to bring all components up to date. The description file can contain up to 1048 description blocks.

## Program Maintenance Utility (NMAKE)



## Special Features

The following are special features of description files and blocks:

- Description files can contain macro definitions and use macros in description blocks. Macros allow easy substitution of one text string for another.
- Description files can contain inference rules. Inference rules allow NMAKE to infer which commands to execute based on the filename extensions used for targets and dependents.

- You can specify directories for NMAKE to search for dependent files by using the following syntax:

```
targets : {directory1;directory2...}dependents
```

NMAKE searches the current directory first, then *directory1*, *directory2*, and so on.

- A command can be placed on the same line as the target and dependent files by using a semicolon (;) as depicted below:

```
targets... : dependents... ; command
```

- A long command can span several lines if each line ends with a backslash (\):

```
command \  
  continuation of command
```

- The execution of a command can be modified if you precede the command with special characters.
- If you do not specify a command in a description block, NMAKE looks for an inference rule to build the target.

## Program Maintenance Utility (NMAKE)

- Wild card characters (\* and ?) can be used in description blocks. For example, the following description block compiles all source files with the .c extension:  

```
astro.exe : *.c
    gcc **
```
- NMAKE will expand the \*.c specification into the complete list of C files in the current directory. \*\* is a complete list of dependents specified for the current target.
- NMAKE uses several punctuation characters in its syntax. To use one of these characters as a literal character, place an escape character ( ^ ) in front of it. For a list of punctuation characters, see “Escape Characters” on page 733.
- Normally a target file can appear in only one description block. A special syntax allows you to use a target in several description blocks.
- A special syntax allows you to determine the drive, path, base name, and extension of the first dependent file in a description block.

### Targets In Several Description Blocks

Using a file as a target in more than one description block causes NMAKE to end. You can overcome this limitation by using two colons (::) as the target/dependent separator instead of one colon.

The following description block is permissible:

```
X :: A
    command
X :: B
    command
```

The following causes NMAKE to end:

```
X : A
    command
X : B
    command
```

It is permissible to use single colons if the target/dependent lines are grouped above the same commands. The following is permissible:

```
X : A
X : B
    command
```

## Program Maintenance Utility (NMAKE)

### Double Colon (::) Target/Dependent Separator Example

```
target.lib :: a.asm b.asm c.asm
ml a.asm b.asm c.asm
ilibr target a.obj b.obj c.obj

target.lib :: d.c e.c
icc /c d.c e.c
ilibr target d.obj e.obj
```

These two description blocks both update the library named `target.lib`. If any of the assembly-language files have changed more recently than the library file, NMAKE executes the commands in the first block to assemble the source files and update the library. Similarly, if any of the C-language files have changed, NMAKE executes the second group of commands to compile the C files and update the library.

---

## Macros

Macros provide a convenient way to replace one string with another in the description file. The text is automatically replaced each time NMAKE is run. This feature makes it easy to change text throughout the description file without having to edit every line that uses the text.

Two common uses of macros are:

- To create a standard description file for several projects. The macro represents the file names in commands. These file names are defined when you run NMAKE. When you switch to a different project, changing the macro changes the file names NMAKE uses throughout the description file.
- To control the options that NMAKE passes to the compiler, assembler, or linker. When using a macro to specify the options, you can quickly change the options throughout the description file in one easy step.

A macro can be defined:

- In a description file
- On the command line
- In `TOOLS.INI`
- Through inheritance from environment variables

## Program Maintenance Utility (NMAKE)

### Macros Example



```
program = flash
c = ilink
options =

$(program).exe : $(program).obj
    $c $(options) $(program).obj;
```

The example above defines three macros. The description block executes the following commands:

```
flash.exe : flash.obj
    ilink flash.obj;
```

### Special Features

Macros have the following special features:

- When using a macro, you can substitute text in the macro itself.
- Several macros have been predefined for special purposes.
- If a macro is defined more than once, precedence rules govern which definition is used.
- You can also put macros into your `TOOLS.INI` file.

### Macros in a Description File

Before using a macro, you need to define it, either on the NMAKE command line or in your description file. Description file macro definitions look like this:

```
macroname = macrostring
```

Macro names can be any combination of alphanumeric characters and the underscore character ( `_` ), and they are case-sensitive. A macro string can be any string of characters.

The first character of the macro name must be the first character on the line. NMAKE ignores any spaces before or after the equal sign ( `=` ).

The macro string can be a null string and can contain embedded spaces. Do not enclose the macro string in quotation marks in the description file; quotation marks are used only when you define macros on the command line.

### Macros on the Command Line

Before using a macro, you need to define it, either on the NMAKE command line or in your description file. Command-line macro definitions look like this:

```
macroname=macrostring
```

## Program Maintenance Utility (NMAKE)

No spaces can surround the equal sign. If you embed spaces, NMAKE might misinterpret your macro. If your macro string contains embedded spaces, enclose it in double quotation marks ( " ) like this:

```
macroname="macro string"
```

or simply enclose the entire macro definition in double quotation marks ( " ) like this:

```
"macroname = macro string"
```

Macro names can be any combination of alphanumeric characters and the underscore character ( \_ ), and they are case-sensitive. A macro string can be any string of characters or a null string.

## Inherited Macros

NMAKE *inherits* all current environment variables as macros. For example, if you have a PATH environment variable defined as PATH = C:\TOOLS\BIN, the string C:\TOOLS\BIN is substituted when you use PATH in the description file.

You can redefine inherited macros by including a line such as the example above in a description file. While NMAKE is executing, the macro takes on the redefined definition. When NMAKE terminates, however, the environment variable resumes its original value.

The Override Environment Variables (/E) option disables inherited macro redefinition. If you use this option, NMAKE ignores any attempt to redefine an inherited macro.

**Note:** The macro name, for any macros that you define, is case sensitive. For example, if you create the following macro:

```
UPPER=UpperCase
```

then, \$(UPPER) will return the value, but \$(upper) will not. Inherited macro names (i.e. those created automatically from environment variables) must always be UPPERCASE.

## Defined Macros

After you have defined a macro, you can use it anywhere in your description file with the following syntax:

```
$(macroname)
```

The parentheses are not required if the macro name is only one character long. To use a dollar sign ( \$ ) without using a macro, enter two dollar signs ( \$\$ ), or use the caret ( ^ ) before the dollar sign as an escape character.

## Program Maintenance Utility (NMAKE)

When NMAKE runs, it replaces all occurrences of `$(macroname)` with the defined macro string. If the macro is undefined, nothing is substituted. After a macro is defined, you can cancel it only with the `!UNDEF` directive.

### Macro Substitutions

Just as you use macros to substitute text within a description file, you use the following syntax to substitute text within a macro:

```
$(macroname: string1 = string2)
```

Every occurrence of *string1* is replaced by *string2* in *macroname*. Spaces between the colon and *string1* are considered part of *string1*. If *string2* is a null string, all occurrences of *string1* are deleted from the macro. The colon ( `:` ) must immediately follow *macroname*.

**Note:** The replacement of *string1* with *string2* in the macro is not a permanent change. If you use the macro again without a substitution, you get the original unchanged macro.



#### Example

```
SOURCES = one.c two.c three.c
program.exe : $(SOURCES:.c=.obj)
    ilink **;
```

The example above defines a macro called `SOURCES`, which contains the names of three C source files. With this macro, the target/dependent line substitutes the `.obj` extension for the `.c` extension. Thus, NMAKE executes the following command:

```
ilink one.obj two.obj three.obj;
```

**Note:** `**` is a special macro that translates to all dependent files for a given target.

---

### Special Macros

NMAKE predefines several macros. The first six macros below return one or more file specifications for the files in the target/dependent line of a description block. Except where noted, the file specification includes the path of the file, the base filename, and the filename extension.

Macro	Value
<code>\$@</code>	The specification of the target file.
<code>\$*</code>	The base name (without extension) of the target file. Path information is also returned if the path was specified as part of the target filename. This macro cannot be used in a dependent list.
<code>**</code>	The specifications of the dependent files.

## Program Maintenance Utility (NMAKE)

<b>\$?</b>	The specifications for only those dependent files that are out-of-date with respect to the targets.
<b>\$&lt;</b>	The specification of a single dependent file that is out-of-date with respect to the targets. This macro is used only in inference rules.
<b>\$\$@</b>	The file specification of the target that NMAKE is currently evaluating. This is a dynamic dependency parameter, used only in dependent lists.
<b>\$(CC)</b>	The string ICC, which is the command to run the C Set ++ Compiler. You can redefine this macro to use a different command.
<b>\$(AS)</b>	The string MASM, which is the command to run the Macro Assembler (MASM). You can redefine this macro to use a different command.
<b>\$(MAKE)</b>	The command name used to run NMAKE. This macro is used to invoke NMAKE recursively. If you redefine this macro, NMAKE issues a warning message.  <b>Note:</b> NMAKE executes the command line in which \$(MAKE) appears, even if the display commands (/N) option is on.
<b>\$(MAKEFLAGS)</b>	The NMAKE options currently in effect. You cannot redefine this macro.

**Note:** The special macros **\$\*\*** and **\$\$@** are the only exceptions to the rule that macro names longer than one character must be enclosed in parentheses.

You can append characters to any of the first six macros in this list to modify the meaning of the macro. However, you cannot use macro substitutions in these macros.

### Special Macros Examples



```
trig.lib : sin.obj cos.obj arctan.obj  
!ilib trig.lib $?
```

In the example above, the macro **\$?** represents the names of all dependent files that are out-of-date with respect to the target file. The exclamation point ( **!** ) preceding the **ILIB** command causes NMAKE to execute the **ILIB** command once for each dependent file in the list. As a result of this description, the **ILIB** command is executed up to three times, each time replacing a module with a newer version.



## Program Maintenance Utility (NMAKE)

```
DIR=c:\include
$(DIR)\globals.h : globals.h
    copy globals.h $@
$(DIR)\types.h : types.h
    copy types.h $@
$(DIR)\macros.h : macros.h
    copy macros.h $@
```

The example above shows how to update a group of include files. Each of the files, `globals.h`, `types.h`, and `macros.h`, in the directory `c:\include` depends on its counterpart in the current directory. If one of the include files is out-of-date, NMAKE replaces it with the file of the same name from the current directory.

The following description file, which uses the special macro `$$@`, is equivalent:

```
DIR=c:\include
$(DIR)\globals.h $(DIR)\types.h $(DIR)\macros.h : $$(@F)
!copy $? $@
```

The special macro `$$(@F)` signifies the file name (without the path) of the current target.

When NMAKE evaluates the description block, it evaluates the three targets, one at a time, with respect to their dependents. Thus, NMAKE first checks whether `c:\include\globals.h` is out-of-date compared with `globals.h` in the current directory. If so, it executes the command to copy the dependent file `globals.h` to the target. NMAKE repeats the procedure for the other two targets.

Note that on the command line, the macro `$$?` refers to the dependent for this target. The macro `$$@` specifies the full file specification of the target file.

### File-Specification Parts

A full file specification gives the base name of the file, the file-name extension, and the path. The path provides the disk-drive identifier and the sequence of directories needed to locate the file on the disk.

For example, the file specification

`c:\source\prog\sort.obj`

has the following parts:



Path Name	<code>c:\source\prog</code>
Base File Name	<code>sort</code>
File-Name Extension	<code>.obj</code>

## Program Maintenance Utility (NMAKE)

### Characters That Modify Special Macros

The following six macros all resolve to a file specification (or possibly several file specifications for `$**` and `$?`):

`$*`     `$@`     `**`     `$<`     `$?`     `$$@`

You can append characters to any of these macros to modify the file name returned by the macro. Depending on which character you use, parts of the full file specification are returned:

File Part Returned	Appended Character			
	D	F	B	R
File Path	Yes	No	No	Yes
Base File Name	No	Yes	Yes	Yes
File Name Extension	No	Yes	No	No

### Modified Special Macros Example



If the macro `$@` has the value

`c:\source\prog\sort.obj`

then the following values are returned for the modified macro:

Macro	Value
<code>\$(@D)</code>	<code>c:\source\prog</code>
<code>\$(@F)</code>	<code>sort.obj</code>
<code>\$(@B)</code>	<code>sort</code>
<code>\$(@R)</code>	<code>c:\source\prog\sort</code>

**Note:** Modified macros are always longer than a single character — they must be enclosed by parentheses when used.

### Macro Precedence Rules

When the same macro is defined in more than one place, the definition with the highest priority is used:

Priority	Definition
1 (Highest)	Command line
2	Description file
3	Environment variables
4	TOOLS.INI file
5 (Lowest)	Predefined macros (such as CC and AS)

If you invoke NMAKE with the Overriding Macro Definitions (`/E`) option, macros defined by environment variables take precedence over those defined in a description file.

---

### Inference Rules

Inference rules are templates from which NMAKE infers what to do with a description block when no commands are given. Only those extensions defined in a .SUFFIXES list can have inference rules. The extensions .c, .obj, .asm, and .exe are automatically included in .SUFFIXES.

When NMAKE encounters a description block with no commands, it looks for an inference rule that specifies how to create the target from the dependent files, given the two file extensions. Similarly, if a dependent file does not exist, NMAKE looks for an inference rule that specifies how to create the dependent from another file with the same base name.

NMAKE applies an inference rule only if the base name of the file it is trying to create matches the base name of a file that already exists.

In effect, inference rules are useful only when there is a one-to-one correspondence between the files with the "from" extension and the files with the "to" extension. You cannot, for example, define an inference rule that inserts a number of modules into a library.

The use of inference rules eliminates the need to put the same commands in several description blocks. For example, you can use inference rules to specify a single `icc` command that changes any C source file (with a .c extension) to an object file (with a .obj extension).

You define an inference rule by including text of the following form in your description file or in your `TOOLS.INI` file — see “Special Features”.

```
.fromext.toext:  
commands  
:
```

The elements of the inference rule are:

<i>fromext</i>	The file-name extension for dependent files to build a target
<i>toext</i>	The file-name extension for target files to be built
<i>commands</i>	The commands to build the <i>toext</i> target from the <i>fromext</i> dependent.

For example, an inference rule to convert C source files (with the .c extension) to C object files (with the .obj extension) is

```
.c.obj:  
icc $<
```

## Program Maintenance Utility (NMAKE)

**Note:** The special macro \$< represents the name of a dependent out-of-date relative to the target.

### Special Features

- You can specify a path where NMAKE should look for target and dependent files used in inference rules.
- Inference rules are predefined for compiling and linking C programs, and for assembling programs.
- NMAKE looks for inference rules in the `TOOLS.INI` file if it cannot find a rule in a description file.
- Only those extensions defined in a `.SUFFIXES` list can have inference rules. The extensions `.c`, `.obj`, `.asm`, and `.exe` are automatically included in `.SUFFIXES`.

---

### Inference Rules Example



```
.obj.exe:
    ilink $<;

example1.exe: example1.obj

example2.exe: example2.obj
    ilink /co example2,,libv3.lib
```

The first line above defines an inference rule that causes the `ILINK` command to create an executable file whenever a change is made in the corresponding object file. The file name in the inference rule is specified with the special macro \$< so that the rule applies to any `.obj` file with an out-of-date executable file.

When NMAKE does not find any commands in the first description block, it checks for a rule that might apply and finds the rule defined on the first two lines of the description file. NMAKE applies the rule, replacing \$< with `example1.obj` when it executes the command, so that the `ILINK` command becomes

```
ilink example1.obj;
```

NMAKE does not search for an inference rule when examining the second description block, because a command is explicitly given.

### Inference-Rule Path Specifications

When defining an inference rule, you can indicate to NMAKE where to look for target and dependent files. Use the following syntax:

```
{frompath}.fromext{topath}.toext
commands
:
```

## Program Maintenance Utility (NMAKE)

NMAKE looks in the directory specified by *frompath* for files with the *fromext* extension. It executes the commands to build files with the *toext* extension in the directory specified by *topath*.

### Predefined Inference Rules

NMAKE predefines several inference rules:

Figure 182. NMAKE Predefined Inference Rules

Inference Rule	Command Action	Default
.c.obj	\$(CC) \$(CFLAGS) /c \$*.c	icc /c \$*.c
.c.exe	\$(CC) \$(CFLAGS) \$*.c	icc \$*.c
.asm.obj	\$(AS) \$(AFLAGS) \$*;	masm \$*;

#### Notes:

1. The first two rules automatically compile and link C programs.
2. The last rule automatically assembles programs.
3. The above are the most often used predefined inference rules. For a complete list of predefined inference rules, execute a makefile and specify the /p option. All available inference rules will be displayed.

---

### Directives

Using directives, you can construct description files similar to batch files. NMAKE provides directives that:

- Conditionally execute commands
- Display error messages
- Include the contents of other files
- Turn some NMAKE options on or off

Each directive begins with an exclamation point ( ! ) in the first column of the description file. Spaces can be placed between the exclamation point and the directive keyword.

The list below describes the directives:

**!IF *expression*** Executes the statements between the !IF keyword and the next !ELSE or !ENDIF directive if *expression* evaluates to a nonzero value.

The *expression* used with the !IF directive can consist of integer constants, string constants, or exit codes returned by programs.

## Program Maintenance Utility (NMAKE)


Integer constants can use the C unary operators for numerical negation ( - ), one's complement ( ~ ), and logical negation ( ! ). You can also use any of the C binary operators listed below:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
&	Bitwise AND
	Bitwise OR
^^	Bitwise XOR
&&	Logical AND
	Logical OR
<<	Left shift
>>	Right shift
==	Equality
!=	Inequality
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

### Notes:

1. You can use parentheses to group expressions.
2. Values are assumed to be decimal values unless specified with a leading 0 (octal) or leading 0x (hexadecimal).
3. Strings are enclosed by quotation marks ( " ). You can use the equality ( == ) and inequality ( != ) operators to compare two strings.
4. You can invoke a program in an expression by enclosing the program name in square brackets ( [ ] ). The exit code returned by the program is used in the expression.

## Program Maintenance Utility (NMAKE)

- !ELSE** Executes the statements between the !ELSE and !ENDIF directives if the statements preceding the !ELSE directive were not executed.
- !ENDIF** Marks the end of the !IF, !IFDEF, or !IFNDEF block of statements.
- !IFDEF *macroname***  
Executes the statements between the !IFDEF keyword and the next !ELSE or !ENDIF directive if *macroname* is defined in the description file. If a macro has been defined as null, it is still considered to be defined.
- !IFNDEF *macroname***  
Executes the statements between the !IFNDEF keyword and the next !ELSE or !ENDIF directive if *macroname* is not defined in the description file.
- !UNDEF *macroname***  
Undefines a previously defined macro.
- !ERROR *text*** Prints text and then stops execution.
- !INCLUDE *filename***  
Reads and evaluates the file *filename* before continuing with the current description file. If *filename* is enclosed by angle brackets (<>), NMAKE searches for the file in the directories specified by the INCLUDE macro; otherwise, it looks only in the current directory. The INCLUDE macro is initially set to the value of the INCLUDE environment variable.
- !CMDSWITCHES {+|-}*opt***  
Turns on or off one of four NMAKE options: /D, /I, /N, and /S. If no options are specified, the options are reset to the values they had when NMAKE was started. To turn an option on, precede it with a plus sign (+); to turn it off, precede it with a minus sign (-). This directive updates the MAKEFLAGS macro.
-  See “Special Macros” on page 721.

## Program Maintenance Utility (NMAKE)

### Directives Example



```
!INCLUDE <infrules.txt>
!CMDSWITCHES +D
winner.exe:winner.obj
!IFDEF DEBUG
! IF "$ (DEBUG)"=="y"
    ilink /co winner.obj;
! ELSE
    ilink winner.obj;
! ENDIF
!ELSE
! ERROR Macro named DEBUG is not defined.
!ENDIF
```

The directives in this example do the following:

- The `!INCLUDE` directive causes the file `infrules.txt` to be read and evaluated as if it were part of the description file.
- The `!CMDSWITCHES` directive turns on the `/D` option, which displays the dates of the files as they are checked.
- If `winner.exe` is out-of-date with respect to `winner.obj`, the `!IFDEF` directive checks to see whether the macro `DEBUG` is defined. If it is defined, the `!IF` directive checks to see whether it is set to `y`. If it is, the linker is invoked with the `/CO` option; otherwise, it is invoked without the `/CO`. If the `DEBUG` macro is not defined, the `!ERROR` directive prints the message and NMAKE stops executing.

### Pseudotargets


A *pseudotarget* is a target in a description block that is not a file. Instead, it is a name that serves as a "handle" for building a group of files or executing a group of commands. In the following example, `UPDATE` is a pseudotarget:



```
UPDATE: *.*
!copy *** a:\product
```

When NMAKE evaluates a pseudotarget, it always considers the dependents to be out-of-date. In the description above, NMAKE copies each of the dependent files to the specified drive and directory.

NMAKE predefines several pseudotargets for special purposes.

 See "Predefined Pseudotargets" on page 731.



## Program Maintenance Utility (NMAKE)


### Predefined Pseudotargets

NMAKE predefines several pseudotargets that provide special rules within a description file:

#### **.SILENT Pseudotarget**

**Syntax:** .SILENT : dependents...

This pseudotarget suppresses the display of executed commands for a single description block. The /S option does the same thing for all description blocks.

 See “Suppress Command Display (/S)” on page 715.

#### **.IGNORE Pseudotarget**

**Syntax:** .IGNORE : dependents...

This pseudotarget ignores exit codes returned by programs for a single description block. The /I option does the same thing for all description blocks.

 See “Ignore Exit Codes (/I)” on page 714.

#### **.SUFFIXES Pseudotarget**

**Syntax:** .SUFFIXES : extensions...

This pseudotarget defines file extensions to try when NMAKE needs to build a target file for which no dependents are specified. NMAKE searches the current directory for a file with the same name as the target file and an extension in <extensions...>. If NMAKE finds such a file, and if an inference rule applies to the file, NMAKE treats the file as a dependent of the target.

The .SUFFIXES pseudotarget is predefined as

```
.SUFFIXES : .obj .exe .c .asm
```

To add extensions to the list, specify .SUFFIXES : followed by the new extensions.

To clear the list, specify

```
.SUFFIXES:
```

**Note:** Only those extensions specified in .SUFFIXES can have inference rules.

NMAKE ignores inference rules unless the extensions have been specified in a .SUFFIXES list.

#### **.PRECIOUS Pseudotarget**

**Syntax:** .PRECIOUS : targets...

This pseudotarget tells NMAKE not to delete a target even if the commands that build it are terminated or interrupted. This pseudotarget overrides the NMAKE

## Program Maintenance Utility (NMAKE)

default. By default, NMAKE deletes the target if it cannot be sure that the target was built successfully.



For example,

```
.PRECIOUS : tools.lib
tools.lib : a2z.obj z2a.obj
command
:
```

If the commands to build `tools.lib` are interrupted, leaving an incomplete file, NMAKE does not delete the partially built `tools.lib`.

**Note:** The pseudotarget `.PRECIOUS` is useful only in limited circumstances. Most professional development tools have their own interrupt handlers and "clean up" when errors occur.

---

## Inline Files

You may need to issue a command in the description file with a list of arguments exceeding the command-line limit of the operating system. Just as NMAKE supports the use of command files, it can also generate inline files which are read as response files by other programs.

To generate an inline file, use the following syntax for your description block:

```
target : dependents
  command @<<[filename]
inline file text
<< [KEEP | NOKEEP]
```

All of the text between the two sets of double less than signs (<<) is placed into an inline file and given the name *filename*. You can refer to the inline file at a later time by using *filename*. If *filename* is not given, NMAKE gives the file a unique name in the directory specified by the TMP environment variable if it is defined. Otherwise, NMAKE creates a unique file name in the current directory.

The inline file can be temporary or permanent. If you do not specify otherwise, or if you specify the keyword NOKEEP, the inline file is temporary. Specify KEEP to retain the file.

**Note:** The at sign (@) is not part of the NMAKE syntax but is the typical character used by utilities (such as LINK386) to designate a file as a response file.

## Program Maintenance Utility (NMAKE)

### Inline Files Example



```
math.lib : add.obj sub.obj mul.obj div.obj
    ilib @<<
math.lib
add.obj sub.obj mul.obj div.obj
/L:listing
<<
```

The above example creates an inline file and uses it to invoke the Library Manager (ILIB). The inline file is used as a response file by (ILIB). It specifies which library to use, the commands to execute, and the listing file to produce. The inline file contains the following:

```
math.lib
add.obj sub.obj mul.obj div.obj
/L:listing
```

Because no file name is listed after the ILIB command, the inline file is given a unique name and placed into the current directory (or the directory defined by the TMP environment variable).

### Escape Characters

NMAKE uses the following punctuation characters in its syntax:

(	)	#	\$	^	\
{	}	!	@	-	

To use one of these characters in a command and not have it interpreted by NMAKE, use a caret ( ^ ) in front of the character.

For example,

BIG^#.C

is treated as

BIG#.C

With the caret, you can include a literal newline character in a description file. This capability is useful in macro definitions, as in the following example:

```
XYZ=abc^<ENTER>
def
```

The effect is equivalent to the effect of assigning the C-style string `abc\ndef` to the XYZ macro. Note that this effect differs from the effect of using the backslash ( \ ) to continue a line. A newline character that follows a backslash is replaced with a space.

## Program Maintenance Utility (NMAKE)

NMAKE ignores a caret that is not followed by any of the characters it uses in its syntax. A caret that appears within quotation marks is not treated as an escape character.

**Note:** The escape character cannot be used in the command portion of a dependency block.

---

### Characters That Modify Commands

Any of three characters can be placed in front of a command to modify how the command is run:

- (dash)** Turns off error checking for the command
- @ (at sign)** Suppresses display of the command
- ! (exclamation point)** Executes the command for each dependent file

#### NOTE

1. Spaces can separate the modifying character from the command. Any command on a separate line — whether modified or not — must be indented by one or more spaces or tabs.
2. You can use more than one character to modify a single command.

### Turn Error Checking Off (-)

**Syntax:** `-[n] command`

The `/I` option globally turns command error-checking off. The dash (`-`) command modifier overrides the global setting to turn error checking off for commands individually. This modifier is used in two ways:

- A dash without a number turns off all error checking.
- A dash followed by a number causes NMAKE to abort only if the exit code returned by the command is greater than the number.

 See “Ignore Exit Codes (`/I`)” on page 714.

### Dash Command Modifier Examples



```
light.lst : light.txt
- flash light.txt
```

In the example above, NMAKE never ends, regardless of the exit code returned by `flash`.

## Program Maintenance Utility (NMAKE)

```
light.lst : light.txt
-l flash light.txt
```


In the example above, NMAKE ends if the exit code returned by `flash` is greater than 1.

### Suppress Command Display (@)

**Syntax:** @ command

The `/S` option globally suppresses the display of commands while NMAKE is running. The at sign (@) modifier suppresses the display for individual commands.

**Note:** Regardless of the `/S` option or the @ modifier, output generated by the command itself always appears.

 See “Suppress Command Display (/S)” on page 715.

### At Sign (@) Command Modifier Example



Suppress Command Display (@)

```
sort.exe:sort.obj
@ echo sorting
```

The command line calling the `echo` command is not displayed. The output of the `echo` command, however, is displayed.

### Execute Command for Dependents (!)

**Syntax:** ! command

The exclamation-point command modifier causes the command to be executed for each dependent file if the command uses one of the special macros `$?` or `$**`. The `$?` macro refers to all dependent files out-of-date with respect to the target. The `$**` macro refers to all dependent files in the description block.

 See “Special Macros” on page 721.

### Exclamation Point (!) Command Modifier Examples



```
leap.txt : hop.asm skip.bas jump.c
! print $** lpt1:
```

The example above executes the following three commands, regardless of the modification dates of the dependent file:

## Program Maintenance Utility (NMAKE)

```
print hop.asm lpt1:
print skip.bas lpt1:
print jump.c lpt1:

leap.txt : hop.asm skip.bas jump.c
! print $? lpt1:
```

The example above executes the print command only for those dependent files with modification dates later than that of the leap.txt file. If hop.asm and jump.c have modification dates later than leap.txt, the following two commands are executed:

```
print hop.asm lpt1:
print jump.c lpt1:
```

## EXTMAKE Syntax

Description files can use a special syntax to determine the drive, path, base name, and extension of the first dependent file in a description block. This syntax is called the *extmake* syntax.

The characters %s represent the complete file specification of the first dependent file. Various parts of the file specification are represented using the syntax

%|partsF

where <parts> is a combination of the following letters:

- d** Drive
- p** Path
- f** Base name
- e** Extension

For example, to specify the drive and path name of the first dependent file in a description block, use:

%|dpF

The percent symbol (%) is a replacement in DOS, Windows, and OS/2 command lines. To use the percent symbol in command-line arguments, use a double percent (%%).

## Program Maintenance Utility (NMAKE)

---

### Macros and Inference Rules in TOOLS.INI

You can place either macros or inference rules in your TOOLS.INI file. NMAKE looks for the TOOLS.INI file first in the current directory and then in the directory indicated by the INIT environment variable.

If NMAKE finds a TOOLS.INI file, it looks for the following tag:

```
[nmake]
```

You can place macros and inference rules below this tag in the same format you would use in a description file.

If a macro or inference rule is defined in both the TOOLS.INI file and the description file, the definition in the description file takes precedence. Also, if you use the /R option, the TOOLS.INI file is ignored.

### TOOLS.INI Example



```
[nmake]
CFLAGS=/ss /ms /Gd-
.c.obj:
    $(CC) -c $(CFLAGS) $*.c
```

These lines in the TOOLS.INI file do the following:

- Define the CFLAGS macro as "/ss /ms /Gd-".
- Redefine the predefined inference rule to build .obj files from .c source files.

## **Program Maintenance Utility (NMAKE)**





## Chapter 64. Using the Message Compiler

The message compiler (`mc.exe`) reads message text files and converts them into binary resource files. These binary resource files can be input to the resource compiler (`irc.exe`) which will generate a `.res` file which can be bound to an application or DLL using the VisualAge for C++ linker (`ilink.exe`).

The format of the input message text file supports specifying multiple versions of the same message text, one for each language supported. It also supports the automatic assignment of code numbers to each message, along with the generation of a C language include file for use by the application for accessing the messages using symbolic constants.

**Note:** If you are coming from the OS/2 platform, you should be aware that the message compiler will generate an include (`.h`) file with the message names defined. If you already have a `.h` file, you may want to rename it to avoid having the message compiler overwrite it with the new `.h` file.

---

### MC Syntax

You can specify all the input the message compiler needs on the command line. Input arguments may be specified in any order.

The syntax of the message compiler (`mc.exe`) command line is:

```
mc [options] filename[.mc]
```

where:

Option	Description
<b>-v</b>	Generates verbose output to stderr
<b>-w</b>	Generates a warning message whenever an insert escape sequence is seen that is a superset of the type supported by OS/2 <code>mkmsgf</code> (i.e. anything other than <code>%0</code> and <code>%n</code> ). This is useful when converting old OS/2 message files to this format.
<b>-s</b>	Adds an extra line to the beginning of each message that is the symbolic name associated with the message ID
<b>-h DirSpec</b>	Specifies the target directory of the generated <code>.h</code> file. The file name is the name of the <code>.mc</code> file with a <code>.h</code> extension.
<b>-r DirSpec</b>	Specifies the target directory of the generated <code>.rc</code> file. The file name is the name of the <code>.mc</code> file with a <code>.rc</code> extension.

## Using the Message Compiler

*filename.mc* Specifies one or more input message files that will be compiled into one or more binary resource files, one for each language that the message files contain message text for

The message compiler reads the .mc file and generates a .h file containing all the symbolic name definitions. For each LanguageId that was used to specify message text, it outputs a binary file containing a message table resource. It also outputs a single resource script (.rc) file that contains the appropriate RC syntax to include each binary file output as a resource with the appropriate name and type IDs.

---

## Input Message File

The input message file is a standard ASCII text file. The purpose of the message text file is to define all of the messages needed by an application in a format that makes it easy to support multiple languages with the same source file.

The basic syntax of the message text file is:

Keyword=Value

where, spaces around the equal sign are ignored and the value is delimited by white space from the next Keyword=Value pair. Case is ignored when comparing against keyword names. The value portion can either be a numeric integer constant, {NUMBER}, using C syntax; a symbol name, {NAME}, that complies with the rules for C identifier names; or a file name that conforms to the rules for the FAT file system (i.e. 8 characters or less, no periods).

Comment lines are allowed in the input message file. Comments must begin with a semicolon (;) in the first column and are terminated by the end of the line. Comments that exist by themselves on a line are copied, as-is, to the output .h file, with the semicolon converted to the C end-of-line comment syntax (//).

The overall structure of the message text file consists of a header section, followed by zero or more message definitions.

## Header Section

The header section contains zero or more of the following keywords:

```
MessageIdTypedef={NAME}  
SeverityNames=({NAME}={NUMBER}:{NAME})  
FacilityNames=({NAME}={NUMBER}:{NAME})  
LanguageNames=({NAME}={NUMBER}:{FILENAME})
```

These keywords have the following meaning:

## Using the Message Compiler

Keyword	Description
<b>MessageIdTypedef</b>	Gives a symbolic name that is output as the typedef name for each numeric MessageId value. The default value is NULL, meaning that there will be no type cast output when defining symbolic names for a MessageId.
<b>SeverityNames</b>	<p>Defines the set of names that are allowed as the value of the Severity keyword in the message definition. The set is delimited by left and right parentheses. Associated with each severity name is a number that, when shifted left by 30, gives the bit pattern to logically OR with the Facility value and MessageId to come up with the full 32-bit message code. The default value is:</p> <pre>SeverityNames=(Success=0x0                 Informational=0x1                 Warning=0x2                 Error=0x3                 )</pre> <p>Severity values occupy the two high bits of a 32-bit message code. Any severity value that does not fit in two bits is an error. The severity codes can be given symbolic names by following each value with :{NAME}.</p>
<b>FacilityNames</b>	<p>Defines the set of names that are allowed as the value of the Facility keyword in the message definition. The set is delimited by left and right parentheses. Associated with each facility name is a number that, when shifted left by 16 bits, gives the bit pattern to logically OR with the Severity value and MessageId to come up with the full 32-bit message code. The default value of this keyword is:</p> <pre>FacilityNames=(System=0xFF                 Application=0xFFF                 )</pre> <p>Facility codes occupy the low order 12 bits of the high order 16 bits of a 32-bit message code. Any facility code that does not fit in 12 bits is an error. This allows for 4096 facility codes. The first 256 are reserved for use by the system software.</p> <p>The facility codes can be given symbolic names by following each value with :{NAME}.</p>
<b>LanguageNames</b>	Defines the set of names that are allowed as the value of the Language keyword in the message definition. The set is delimited by left and right parentheses. Associated with each

## Using the Message Compiler

language name is a number and a file name that will be used to name the binary output file that will contain all of the message text for that language. The number corresponds to the Language ID tag to use in the resource table. The number is separated from the file name with a colon. The initial value of this keyword is:

```
LanguageNames=(English=1:MSG00001)
```

Any new names that an application defines in its .mc file which don't override any of the built-in names will be added to the list of valid languages. This allows an application to support private languages with descriptive names.

## Message Definitions

Following the header section of the input message file are zero or more message definitions. Each message definition begins with one or more of the following keywords.

```
MessageId={ | {NUMBER} | +{NUMBER} }
Severity={SEVERITY_NAME}
Facility={FACILITY_NAME}
SymbolicName={NAME}
```

The MessageId keyword is required to mark the beginning of the message definition, although its value is optional. If no value is specified, then the value used will be the last value used for the facility, plus one. If the value is specified as +{NUMBER}, then the value used will be the last value used for the facility plus the number after the plus sign. Otherwise, if a numeric value is given, then that value will be used. Any MessageId value that does not fit in 16 bits is an error.

Severity and Facility are optional keywords that can specify additional bits to OR into the final 32-bit message code. If either of these are not specified, they default to the value last specified for a message definition. The initial values of these prior to processing the first message definition is:

```
Severity=Success
Facility=Application
```

The value associated with these keywords must match one of the names given to the FacilityNames and SeverityNames keywords. The SymbolicName keyword allows the ISV to associate a C symbolic constant name with the final 32-bit message code that is a result of OR-ing together the MessageId, Severity and Facility bits. The constant definition is output to the generated include file with the following format:

```
// message text
#define CONSTANT_SYMBOL_NAME ((MessageIdTypedef) 0x12345678)
```

## Using the Message Compiler

The comment before the definition is a copy of the message text for the first language specified in the message definition. The `CONSTANT_SYMBOL_NAME` is the value of the `SymbolicName` keyword. The `MessageIdTypedef` is the value of the `MessageIdTypedef` keyword. The `MessageIdTypedef` is not output if it is `NULL` (the default value).

The following comment appears in the header of each include file to explain the bit-fields in the 32-bit message:

```
// Values are 32 bit values, laid out as follows:
//
// 3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
// 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
// +---+---+---+---+---+---+---+---+---+---+
// |Sev|C|R|      Facility      |      Code      |
// +---+---+---+---+---+---+---+---+---+---+
//
// where
//
//     Sev - is the severity code
//
//         00 - Success
//         01 - Informational
//         10 - Warning
//         11 - Error
//
//     C - is the Customer code flag
//
//     R - is a reserved bit
//
//     Facility - is the facility code
//
//     Code - is the facility's status code
//
// Define the facility codes
//
//
//
// Define the severity codes
//
```

### Language-Specific Message Text Definitions

One or more message text definitions come after the message definition keywords. Each message text definition begins with the `Language` keyword that identifies which binary output file this message text is to be output to. Beginning on the next line is the first line of the message text. The message text is terminated by a line containing a single period at the beginning of the line, immediately followed by a new line. No spaces are allowed around the keyword. Within the message text, blank lines and white space are preserved as part of the message.

The following is the message text definition format:

## Using the Message Compiler

```
Language={LANGUAGE_NAME}  
{MESSAGETEXT}  
.
```

Within the message text, several escape sequences are supported for dynamically formatting the message. The percent sign character (%) begins all escape sequences.

### Escape Sequence

### Definition

**%0**

This terminates a message text line without a trailing newline. This can be used to build up long lines or to terminate the message itself without a trailing newline, which is useful for prompt messages.

**%n!printf format string!**

This identifies an insert. The value of *n* can be between 1 and 99. The printf format string must be enclosed by exclamation marks. It is optional and defaults to !s! if not specified.

The printf format string can contain the \* specifier for either the precision or width components, and if so, they will consume inserts %n+1 and %n+2 for their values at run time. mc.exe will print a warning message if an explicit reference is made to these inserts elsewhere in the message text.

Inserts must reference a parameter passed to the FormatMessage API call. It will return an error if a message refers to an insert that was not passed to the FormatMessage API call.

Any other character following a percent sign, other than a digit, will be formatted in the output message without the percent sign. For example:

- |           |                                                                                                                                                                                   |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>%%</b> | Will output a single percent sign in the formatted message text.                                                                                                                  |
| <b>%n</b> | Will output a hard line break when it occurs at the end of a line. This is useful when FormatMessage is supplying normal line breaks so that the message fits in a certain width. |
| <b>%r</b> | Will output a hard carriage return, without a trailing newline.                                                                                                                   |
| <b>%b</b> | Will output a space in the formatted message text. This can be used to insure that there are the appropriate number of trailing spaces in a message text line.                    |
| <b>%t</b> | Will output a tab in the formatted message text.                                                                                                                                  |

## Using the Message Compiler

- %.** Will output a single period in the formatted message text. This can be used to get a single period at the beginning of a line without terminating the message text definition.
- %!** Will output a single exclamation point in the formatted message text. This can be used to get an exclamation point immediately after an insert without it being mistaken for the beginning of a printf format string.

**Note:** Unicode is not supported. You may use DBCS in your message text file, as long as you have a text editor that can do this.

### Input Message File Example



Following is an example of an input message file:

```
MessageIdTypeDef=DWORD
LanguageNames=(English=1:MSG00001)
SeverityNames=(Success=0)
FacilityNames=(Application=0x0)

MessageId = 1
Severity = Success
Facility = Application
Language = English
Error : unrecognized command %1
.

MessageId = 2
Severity = Success
Facility = Application
Language = English
Warning : running low on memory
.

MessageId = 3
Severity = Success
Facility = Application
Language = English
Informational : %1 files are now open
.
```

## Using the Message Compiler

---

### Message Win32 API Calls

```
DWORD
APIENTRY
FormatMessage(
    DWORD dwFlags,
    LPVOID lpSource,
    DWORD dwMessageId,
    DWORD dwLanguageId,
    LPSTR lpBuffer,
    DWORD nSize,
    va_list Arguments
)
```

#### Description

This function formats a message string. Input to the function is a message definition. It can come from a buffer passed into this function. It can also come from a message table resource in a module already loaded. Or, the caller can ask this function to search the system message table resource(s) for the message. This function finds the message definition based on the Message ID and the Language ID and copies the message text to the output buffer, processing any imbedded insert sequences if requested.

#### Parameters

##### dwFlags

dwFlags specifies options to the formatting process along with how to interpret the lpSource parameter. The low order 16 bits of this parameter are the maximum width of the line, in characters. Possible values are:

##### **FORMAT\_MESSAGE\_ALLOCATE\_BUFFER**

lpBuffer is a PVOID \* and nSize is the minimum size to allocate. This function will then allocate a buffer large enough to hold the formatted message and store the pointer to the buffer in the location pointed to by lpBuffer. The caller should free the buffer with LocalFree when they are done using it.

##### **FORMAT\_MESSAGE\_IGNORE\_INSERTS**

Insert sequences in the message definition will be ignored and passed through to the output buffer, as is. This is useful for fetching a message for later formatting. If this flag is set, the lpArguments parameter is ignored.

##### **FORMAT\_MESSAGE\_FROM\_STRING**

lpSource is a pointer to a null terminated message definition. It can contain insert sequences, just as the message text in the .mc file can.



## Using the Message Compiler

### **FORMAT\_MESSAGE\_FROM\_HMODULE**

lpSource is a module handle that contains the message table resource(s) to search. If this handle is NULL, then the current process' application image file will be searched.

### **FORMAT\_MESSAGE\_FROM\_SYSTEM**

If the requested message was not found in lpSource, or if lpSource was not examined (i.e. neither of the preceding two flags was specified), then this function will search the system message table resource(s).

### **FORMAT\_MESSAGE\_ARGUMENT\_ARRAY**

If set, this parameter specifies that the passed Arguments parameter is NOT a va\_list structure, but instead is just a pointer to an array of 32-bit values that represent the arguments.

### **FORMAT\_MESSAGE\_MAX\_WIDTH\_MASK**

The low order 8 bits specify the maximum width of each line formatted into the output buffer. A maximum width of zero means that no restrictions are placed on the width, and only the line breaks in the message definition will be placed in the output buffer. If a non-zero value is specified, then line breaks in the message definition text are ignored, and instead line breaks are calculated based on the maximum width, with white space delimited strings never being split across a line break. Hard coded line breaks in the message definition text, that are identified by the %n escape sequence, are always output to the output buffer.

If the width specified is

FORMAT\_MESSAGE\_MAX\_WIDTH\_MASK, then line breaks in the message file are ignored and only hard coded line breaks are kept. None are generated.

### **lpSource**

Specifies where to retrieve the message definition from. The type of this parameter depends upon the settings in the dwFlags parameter.

FORMAT\_MESSAGE\_FROM\_HMODULE - lpSource is an hModule of the module that contains the message table to search.

FORMAT\_MESSAGE\_FROM\_STRING - lpSource is an LPSTR that points to unformatted message text. It will be scanned for inserts and formatted accordingly.

## Using the Message Compiler

If neither of these options is specified, then this parameter is ignored.

<b>dwMessageId</b>	Specifies the 32-bit message identifier that identifies the message being requested. This parameter is ignored if the <code>FORMAT_MESSAGE_FROM_STRING</code> flag is specified.
<b>dwLanguageId</b>	Specifies the 32-bit language identifier that identifies the language of the message being requested. This parameter is ignored if the <code>FORMAT_MESSAGE_FROM_STRING</code> flag is specified.
<b>lpBuffer</b>	Specifies a pointer to a buffer where the formatted message is to be written. A terminating null byte will also be written. If the <code>FORMAT_MESSAGE_ALLOCATE_BUFFER</code> flag was specified, then this parameter points to a 32-bit pointer value that is filled in by this call with a pointer allocated via <code>LocalAlloc</code> to contain the text of the message.
<b>nSize</b>	Specifies the maximum number of bytes that can be stored in the output buffer. This parameter is ignored if the <code>FORMAT_MESSAGE_ALLOCATE_BUFFER</code> flag is set.
<b>Arguments</b>	Specifies a pointer to a variable number of arguments. These arguments are used to satisfy insert requests in the format string. Thus, <code>%1</code> in the format string specifies the first argument in the variable number of arguments described by the <code>Arguments</code> parameter; <code>%3</code> would specify the third, and so on.

The interpretation of each 32-bit arguments value depends on the formatting information associated with the insert in the message definition. The default is to treat each pointer as a pointer to a null terminated string.

By default, the `Arguments` parameter is of type `va_list`, which is a language and implementation specific data type for describing a variable number of arguments. If you do not have a pointer of type `va_list`, then specify the `FORMAT_MESSAGE_ARGUMENT_ARRAY` flag and pass a pointer to an array of 32-bit values that are input to the message formatted as the insert values.

## Using the Message Compiler

### Return Value

FormatMessage returns a value, of type DWORD, which contains the number of bytes actually stored in the output buffer, excluding the terminating null character. It returns 0 if an error has occurred. Extended error status is available via the GetLastError API.

## Using the Message Compiler

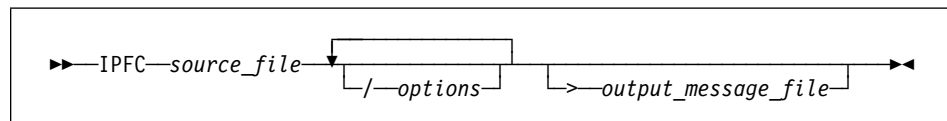


## Chapter 65. Creating Online Documentation

The Information Presentation Facility (IPF) is a tool that you can use to create online information, to specify how it will appear on the screen, to connect various parts of the information, and to provide help information that can be requested by the user. IPF features include:

- A tagging language that formats text and provides ways to connect information and customize the information display.
- A compiler that creates online documents and help windows.
- A viewing program that displays formatted online documents (*iview*).

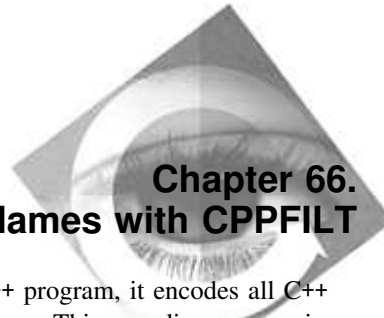
The syntax for the IPF compiler command is:



Enabling help for applications requires programming code that communicates with IPF and with the Windows APIs to display help windows.

For more information on creating help for applications, see the online *Information Presentation Facility Programmer's Guide* in the VisualAge for C++ Information folder.

## **Creating Online Documentation**



## Demangling Compiled C++ Names with CPPFILT

When the VisualAge for C++ compiler compiles a C++ program, it encodes all C++ symbolic names to include type and scoping information. This encoding process is called *mangling*. The linker uses the mangled names in the object files to resolve external references using the exported names.

Tools that use the files with mangled names do not have access to the original source code names, and therefore present the mangled names in their output. Using a process called *demangling*, the CPPFILT utility converts the mangled names to their original source code names so that they can be easily identified.

**Note:** The demangling routines in the runtime library provided with VisualAge for C++ offer another method for converting mangled names to their original source code names. You can use these routines to develop tools that manipulate mangled names. For more information on the demangling routines, refer to the appendix on mapping in the *Programming Guide*, and the information in the `<demangle.h>` header file.

---

### Using the CPPFILT Utility

The CPPFILT utility converts mangled names to demangled names in two separate modes:

- |               |                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Text</b>   | Specify the names of one or more ASCII text files to have the CPPFILT substitute demangled names for mangled names wherever it finds them in the text. Input can also be read from <b>stdin</b> . |
| <b>Binary</b> | Specify the names of one or more object (.obj) and library (.lib) files to produce demangled output in a format that is suitable for inclusion in a DLL module definition (.def) file.            |

All output is sent to **stdout**.

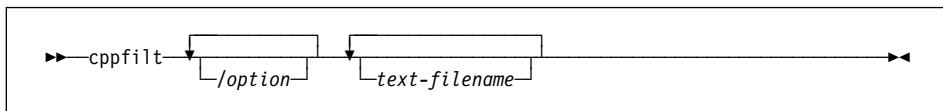
---

### Text Mode

Use the CPPFILT utility in Text mode when you want to simply substitute demangled names for mangled names wherever they are found in the text.

To use the CPPFILT utility in text mode, type `cppfilt` followed by any valid text mode options on the command line.

The syntax for the `cppfilt` command in Text mode is:



## Text Files

The file you specify for the `cppfilt` command in text mode should be an ASCII text file that is present in the current directory, unless you specify its path.

If you do not specify a text file, the input is read from **stdin**.

## Output in Text Mode

Output in text mode would be the input text with the mangled names replaced by their demangled names wherever they are found in the text.

All output is sent to **stdout**.

---

## Text Mode Options

The CPPFILT utility has the following options in Text mode:

- /C** Demangle stand-alone class names
- /H or /?** Display help for CPPFILT
- /M** Produce symbol map
- /Q** Suppress display of logo
- /S** Demangle compiler-generated symbol names
- /T** Produce side-by-side demangling
- /Wnnn** Specify width of field for demangled names

**Note:** You can specify options using the slash form (/S) or the dash form (-S).

## /C (Class Names)

### Syntax:

/C

### Default:

Do not demangle stand-alone class names

Use `/C` to instruct the CPPFILT utility to demangle stand-alone class names.

Stand-alone class names are names that do not appear within the context of a function name or member variable. These names are not normally produced by the compiler.

For example, the stand-alone class name `Q2_1X1Y` would be demangled as `X::Y` if you specify the `/C` option.



If you do not specify the `/C` option, the default is not to demangle stand-alone class names.

## **/H (Help)**

**Syntax:**

`/H`  
`/?`

**Default:**

None

Specify the `/H` or `/?` option on the `CPPFILT` command line to see a short online help on the `cppfilt` command syntax and options.

If you do not specify this option, the default is not to display any online help.

## **/M (Symbol Map)**

**Syntax:**

`/M`

**Default:**

Do not produce symbol map

Specify the `/M` option on the `CPPFILT` command line to produce a symbol map on standard output. The map contains a list of the mangled names and their corresponding demangled names. This output follows the normal filtered output.

If you do not specify the `/M` option, the default is not to produce a symbol map.

## **/Q (Do Not Display Logo)**

**Syntax:**

`/Q`

**Default:**

Display logo and copyright notice

Specify the `/Q` option on the `CPPFILT` command line to suppress the display of the logo and copyright notice for the `CPPFILT` utility.

If you do not specify this option, the default is to display the logo and copyright notice.

## **/S (Special Symbol Names)**

**Syntax:**

`/S`

**Default:**

Do not demangle special  
compiler-generated symbol names

Specify the `/S` option on the CPPFILT command line to instruct the CPPFILT utility to demangle special compiler-generated symbol names.

For example, the compiler-generated symbol name, `__ct__3FooFUi`, that represents a constructor for the class `Foo` would be demangled as `Foo::Foo(unsigned int)`.

If you do not specify the `/S` option, the default is not to demangle special compiler-generated symbol names.

## **/T (Mangled and Demangled Names Together)**

**Syntax:**

`/T`

**Default:**

Replace mangled name with demangled  
name only

Specify the `/T` option on the CPPFILT command line to produce side-by-side demangling. That is, each mangled name is replaced with the demangled name followed by the original mangled name in the text.

If you do not specify the `/T` option, the default is to replace the mangled name by the demangled name only.

## **/Wnnn (Width)**

**Syntax:**

`/Wwidth`

**Default:**

No fixed field width

Specify the `/Wwidth` option on the CPPFILT command line to have the CPPFILT utility print the demangled names in fields *width* characters wide. If the name is shorter than *width*, it is padded to the right with blanks; if longer, it is truncated to *width* characters.

If you do not specify the `/W width` option, the default is not to have a fixed field width when printing demangled names in the text.

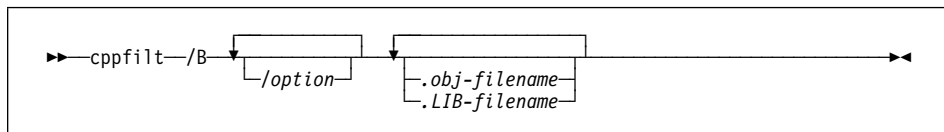
---

## Binary Mode

Use the CPPFILT utility in Binary mode when you want to demangle names in object (.obj) and library (.lib) files to produce output that is suitable for inclusion in a DLL module definition (.def) file.

To use the CPPFILT utility in Binary mode, type `cppfilt /B` followed by any valid Binary mode options on the command line. The CPPFILT utility switches to the Binary mode of operation when it encounters the `/B` option.

The syntax for the `cppfilt` command in Binary mode is:



## .obj and .lib Files

The file names that you specify after the `cppfilt /B` binary-mode command must be object (.obj) or library (.lib) files. They must be present in the current directory, unless their paths are specified. CPPFILT will also search for library files along the paths specified in the LIB environment variable if the files are not found in the current directory.

Input cannot be read from **stdin** in binary mode. You must specify object or library filenames for input.

All output is sent to **stdout**.

## Output in Binary Mode

CPPFILT output in Binary mode lists any libraries and any object files within each library. If you specify the `/X`, `/R`, and `/P` options, the exported, referenced, and public symbols are also listed for each object file.



For example, the command

```
cppfilt /B /P /O 1000 /N c:\ibmcpp\lib\dde4cc.lib
```

would produce the following output for a library file called DDE4CC.LIB:

```
;From library: c:\ibmcpp\lib\dde4cc.lib
;From object file: C:\ibmcpp\src\IILNSEQ.C
;PUBDEFS (Symbols available from object file):
;ILinkedSequenceImpl::setToPrevious(ILinkedSequenceImpl::Node*&) const
setToPrevious__19ILinkedSequenceImplCFRPQ2_19ILinkedSequenceImpl4Node @1000 NONAME
;ILinkedSequenceImpl::allElementsDo(void*,void*) const
allElementsDo__19ILinkedSequenceImplCFPvT1 @1001 NONAME
;ILinkedSequenceImpl::isConsistent() const
isConsistent__19ILinkedSequenceImplCFv @1002 NONAME
;ILinkedSequenceImpl::setToNext(ILinkedSequenceImpl::Node*&) const
setToNext__19ILinkedSequenceImplCFRPQ2_19ILinkedSequenceImpl4Node @1003 NONAME
;ILinkedSequenceImpl::addAsNext(ILinkedSequenceImpl::Node*,ILinkedSequenceImpl::Node*)
addAsNext__19ILinkedSequenceImplFPQ2_19ILinkedSequenceImpl4NodeT1 @1004 NONAME
;From object file: C:\ibmcpp\src\IITBSEQ.C
;PUBDEFS (Symbols available from object file):
;ITabularSequenceImpl::setToPrevious(ITabularSequenceImpl::Cursor&) const
setToPrevious__20ITabularSequenceImplCFRPQ2_20ITabularSequenceImpl6Cursor @1034 NONAME
;ITabularSequenceImpl::allElementsDo(void*)
allElementsDo__20ITabularSequenceImplFPv @1035 NONAME
;ITabularSequenceImpl::removeAll(void*,void*)
removeAll__20ITabularSequenceImplFPvT1 @1036 NONAME
;ITabularSequenceImpl::addAllFrom(const ITabularSequenceImpl&)
addAllFrom__20ITabularSequenceImplFRC20ITabularSequenceImpl @1037 NONAME
;From object file: IIAVLKSS.C
;PUBDEFS (Symbols available from object file):
;IAvKeySortedSetImpl::allElementsDo(void*,void*) const
allElementsDo__20IAvKeySortedSetImplCFPvT1 @1080 NONAME
;IAvKeySortedSetImpl::isFirst(const IAvKeySortedSetImpl::Node*) const
isFirst__20IAvKeySortedSetImplCFPCQ2_20IAvKeySortedSetImpl4Node @1081 NONAME
;IAvKeySortedSetImpl::setPosition(unsigned long,IAvKeySortedSetImpl::Node*&) const
setPosition__20IAvKeySortedSetImplCFUIRPQ2_20IAvKeySortedSetImpl4Node @1082 NONAME
;IAvKeySortedSetImpl::locateOrAddElementWithKey(const void*)
locateOrAddElementWithKey__20IAvKeySortedSetImplFPCv @1083 NONAME
```

**Note:** This is only a partial listing of the actual output.

---

## Binary Mode Options

In binary mode, the CPPFILT utility has the following options:

<b>/B</b>	Operate in Binary mode
<b>/C</b>	Demangle stand-alone class names
<b>/H or /?</b>	Display help
<b>/N</b>	Reference exported names by ordinal number
<b>/O[ord]</b>	Generate ordinals after each demangled name
<b>/P</b>	Include public symbols in output
<b>/Q</b>	Suppress display of logo
<b>/R</b>	Include referenced symbols in output
<b>/S</b>	Demangle special compiler-generated symbol names
<b>/U</b>	Removes a leading underscore from demangled names
<b>/X</b>	Include exported symbols in output
<b>/Z</b>	Suppresses comments in the output

**Note:** You can specify options using the slash form (/R) or the dash form (-R).

## **/B (Operate in Binary Mode)**

**Syntax:**

/B

**Default:**

Operate in Text mode

Specify /B on the CPPFILT command line to instruct CPPFILT to operate in Binary mode.

If you do not specify the /B option, CPPFILT will operate in the default Text mode.

## **/H (Help)**

**Syntax:**

/H

/?

**Default:**

None

Specify the /H or /? option on the CPPFILT command line to see a short online help on the cppfilt command syntax and options.

If you do not specify this option, the default is not to display any online help.

## **/N (NONAME Keyword)**

**Syntax:**

/N

**Default:**

Reference exported names by name


Specify the /N option on the CPPFILT command line to instruct CPPFILT to generate the NONAME keyword as consistent with the module definition (.def) file EXPORTS statement syntax. The NONAME keyword indicates that the exported names should be referenced by their ordinals, and not by their names.

**Note:** You should also specify /N together with the /O option to generate ordinals.



For example, if you specify /N /O 1000, the output for a single name would look something like this:

```
;ILinkedSequenceImpl::isConsistent() const  
isConsistent__19ILinkedSequenceImplCFv @1000 NONAME
```

For more information on using module definition files,  see Chapter 46, “Module Definition Files” on page 533.

## /O (Ordinals)

**Syntax:**

`/O[ord]`

**Default:**

Do not generate ordinals

Specify the `/O[ord]` option on the CPPFILT command line to generate ordinals after each demangled name. If the optional numeric parameter *ord* is specified, CPPFILT will generate the ordinals starting from *ord*.


The ordinals are generated along with the `@` character as consistent with the module definition (.def) file EXPORTS statement syntax.



For example, if you specify `/O 1000`, the output for a single name would look something like this:

```
;ILinkedSequenceImpl::isConsistent() const  
isConsistent__19ILinkedSequenceImplCFv @1000
```

If you do not specify the `/O` option, the default is not to generate ordinals after each demangled name.

For more information on using module definition files,  see Chapter 46, “Module Definition Files” on page 533.

## /P (Public Symbols)

**Syntax:**

`/P`

**Default:**

Do not include public symbols in output

Specify the `/P` option on the CPPFILT command line to include all public (PUBDEF, COMDAT, COMDEF) symbols in the output.

**Note:** Only the first occurrence of a symbol found within the COMDAT sections will be included in the output. Subsequent occurrences of the same COMDAT symbol will appear as comments in the output.

If you do not specify this option, the default is not to include public symbols in the output.

**Note:** If you do not specify any of the `/X`, `/R`, or `/P` options, the Binary mode output will include only the library and object names, without any symbol names.

## **/Q (Do Not Display Logo)**

**Syntax:**

/Q

**Default:**

Display logo and copyright notice

Specify the /Q option on the CPPFILT command line to suppress the display of the logo and copyright notice for the CPPFILT utility.

If you do not specify this option, the default is to display the logo and copyright notice.

## **/R (Referenced Symbols)**

**Syntax:**

/R

**Default:**

Do not include referenced symbols in output

Specify the /R option on the CPPFILT command line to include all referenced (EXTDEF) symbols in the output.

If you do not specify this option, the default is not to include referenced symbols in the output.

**Note:** If you do not specify any of the /X, /R, or /P options, the binary-mode output will include only the library and object names, without any symbol names.

## **/S (Special Symbol Names)**

**Syntax:**

/S

**Default:**

Do not demangle special compiler-generated symbol names

Specify the /S option on the CPPFILT command line to instruct the CPPFILT utility to demangle special compiler-generated symbol names.

For example, the compiler-generated symbol name, `__ct__3FooFUi`, that represents a constructor for the class `Foo` would be demangled as `Foo::Foo(unsigned int)`.

If you do not specify the /S option, the default is not to demangle special compiler-generated symbol names.

## **/U (Underscore Removal)**

**Syntax:**

/U

**Default:**

Leave leading underscores in mangled names.

Specify the /U option on the CPPFILT command line to instruct the CPPFILT utility to remove a single leading underscore (if present) from mangled names. If more than one leading underscore is present, only one is removed.

If you do not specify the /U option, any leading underscores on mangled names are left.

## **/X (Exported Symbols)**

**Syntax:**

/X

**Default:**

Do not include exported symbols in output

Specify the /X option on the CPPFILT command line to include all exported (EXPDEF) symbols in the output.

If you do not specify /X, the default is not to include exported symbols in the output.

**Note:** If you do not specify any of the /X, /R, or /P options, the Binary mode output will include only the library and object names, without any symbol names.

## **/Z (Suppresses Comments)**

**Syntax:**

/Z

**Default:**

Leave comments in output

Specify the /Z option on the CPPFILT command line to instruct the CPPFILT utility to suppress all comments in the output.

By default, mangled names are left in the output as comments.





## Chapter 67. Resource Image Conversion Utility

This utility is used to convert bitmap (.BMP), icon (.ICO), pointer (.PTR) or cursor (.CUR) images from an OS/2 format to a Windows format, or vice-versa. You are also provided with the option to save your images as single or multimember images.



A member is a specific size and color derivation of an image. An image can be saved with single or multiple members, with each member having its own size and color specifications of the selected image (for use with different display drivers). The graphical version of this program will allow you to select, or exclude, specific members for your converted image or have chosen members saved as individual files. If you are using this utility, and you need particular size and color specifications for your image, then you should use the graphical version of this utility for your conversions.

This utility cannot convert an image to a different image type; for example, a bitmap cannot be converted to an icon using this utility.

**Note:** A pointer is the OS/2 terminology for a cursor in Windows.

---

### Starting the Utility

Invoke the graphical version of this utility by typing **ibmpcni**, on your command line, in the directory where you installed VisualAge for C++; if you have installed VisualAge for C++ to a custom directory, type this command from the child *ibmcppw* directory, located in the directory you chose during the install. You may also choose to convert your images using the command line version of this utility, to invoke this version type **ibmpcnv**.

**Note:** If you are converting a multimember image you should use the graphical interface for your conversions. The command line version of this utility will not allow you to convert selected image members, rather it will select only the first member of the image and it will not inform you as to the size and color specifications of that selected member.

---

## Using the Utility

Convert your images through either the graphical interface or the command line.

### Converting Your Images Using the Graphical Interface

Complete the following steps to convert your images to the appropriate platform.

1. Under **Options**, select **Image preselection** in order to designate the predefined size and color combinations (called members) that you want to be automatically selected when you open the image that you will be converting. (The default is to preselect all of the available members).
2. From the **File** pull down menu, select **Open** and choose an image that you wish to convert.
3. Choose the platform of the image that you wish to convert to by selecting Windows or OS/2 from the **Format** drop down menu (the default is Windows).
4. Under the **Mode** option, on the action bar, you must specify whether your image should be treated individually or as a group of images.

*Group* In this mode only one name field will be displayed and all of the members, which you select, will be saved to one image. Each member will have a check mark option box beside it; simply de-select, or select, those members that you wish to be included in your final converted image. (The default is for all of the members, specified in **Image Preselection**, to be selected.) Type the pathname and name for the converted image in the **Save to** field provided, or select an existing pathname and name using the **Select Filename** option, under **File**, from the action bar menu.

*Items* This mode allows you to save each member to an individual file. You must specify a different name, in the **Save to** fields, for each member that you wish to be converted to separate files. If you do not specify a name for the member it will not be converted. Type the pathnames and names that you want the converted members to be called in the individual **Save to** fields provided, or select existing pathnames and names using the **Select Filename** option, under **File**, from the action bar menu.

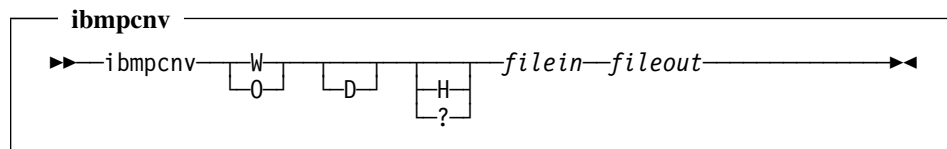
#### Notes:

- a. Since the Windows operating environment does not support multimember bitmap images, the **Group** option is not available for *bitmaps* being converted to Windows; the image will default to the **Items** mode.
- b. In order to use the **Select Filename** option your cursor must have focus on the **Save to** field.

5. Select **Save**, under **File**, from the action bar menu to save your new image(s).  
You must have an entry in the **Save to** field(s) to use this option.

## Converting Your Images Using the Command Line Processor

For conversions from the command line processor use the **ibmpcnv** command as follows:



Where:

- W** Save as Windows resource
- O** Save as OS/2 2.x resource or higher
- D** Convert directory contents
- H** Displays command line help
- ?** Displays command line help
- filein** Input file (if -D, use input directory)
- fileout** Output file (if -D, output directory)

### Notes:

1. These parameters are not case sensitive.
2. A "/" may be used instead of "-" as a delimiter for the parameters.
3. The input and output filenames that you specify, when converting a single file, **must** be different from each other.
4. The **ibmpcnv** utility only allows you to save members as **Item** images, it is not recommended if you need to maintain certain size and color specifications in your converted image.
5. Only .BMP, .PTR, .ICO and .CUR images can be converted with this utility.
6. If the images that you wish to convert are not in the same directory as the installed utility, then you need to specify the pathname along with each of the *filein* and *fileout* name parameters.

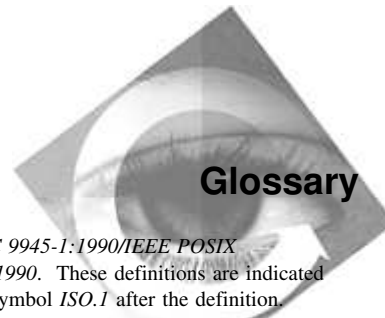
**Examples of Command Line Conversions**

To convert a Windows icon, called river.ico, to an OS/2 format, you would type the following:

```
ibmpcnv -o river.ico os2river.ico
```

To convert an OS/2 directory of icons (in a directory called os2icos), to a Windows format (to be saved in a directory called winicos), you would type the following:

```
ibmpcnv -w -d c:\os2icos c:\winicos
```



This glossary defines terms and abbreviations that are used in this book. Included are terms and definitions from the following sources:

- *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018. Such definitions are indicated by the symbol *ANSI* after the definition.
- *IBM Dictionary of Computing*, SC20-1699. These definitions are indicated by the registered trademark *IBM* after the definition.
- *X/Open CAE Specification. Commands and Utilities, Issue 4. July, 1992*. These definitions are indicated by the symbol *X/Open* after the definition.

- *ISO/IEC 9945-1:1990/IEEE POSIX 1003.1-1990*. These definitions are indicated by the symbol *ISO.1* after the definition.
- *The Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol *ISO-JTC1* after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol *ISO Draft* after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

## A

**abstract class.** (1) A class with at least one pure virtual function that is used as a base class for other classes. The abstract class represents a concept; classes derived from it represent implementations of the concept. You cannot construct an object of an abstract class. See also base class. (2) A class that allows polymorphism.

**abstract data type.** A mathematical model that includes a structure for storing data and operations that can be performed on that data. Common abstract data types include sets, trees, and heaps.

**abstraction (data).** See data abstraction.

**access.** An attribute that determines whether or not a class member is accessible in an expression or declaration. It can be public, protected, or private.

**access declaration.** A declaration used to adjust access to members of a base class.

**access function.** A function that returns information about the elements of an object so that you can analyze various elements of a string.

**access resolution.** The process by which the accessibility of a particular class member is determined.

**access specifier.** One of the C++ keywords public, private, or protected.

**action.** A description of tool or function that can be used to manipulate a project's parts, or build a project's target. Examples are compile, link, and edit.

**action class.** A grouping of actions that perform a similar function.

**actions support DLL.** A dynamic link library that provides such support for an action as determining dependencies and targets if the action is to participate in a build, providing a user interface for setting options, and integrating with the monitor and editor.

## active time •audio track

**active time.** Time when an object, variable, pointer, etc. is on the stack, that is, within scope.

**address space.** (1) The range of addresses available to a computer program. *ANSI.* (2) The complete range of addresses that are available to a programmer. (3) The area of virtual storage available for a particular job. (4) The memory locations that can be referenced by a process. *X/Open. ISO.1.*

**aggregate.** (1) An array or a structure. (2) A compile-time option to show the layout of a structure or union in the listing. (3) An array or a class object with no private or protected members, no constructors, no base classes, and no virtual functions. (4) In programming languages, a structured collection of data items that form a data type. *ISO-JTC1.*

**alignment.** The storing of data in relation to certain machine-dependent boundaries. *IBM.*

**ambiguous derivation.** A derivation where the class is derived from two or more base classes that have members with the same name.

**American National Standards Institute.** See *ANSI.*

**amplifier.** A device that increases the strength of input signals. Also referred to as an amp.

**amplifier-mixer.** A combination amplifier and mixer that is used to control the characters of an audio signal from one or more audio sources. Also referred to as an amp-mixer.

**angle brackets.** The characters < (left angle bracket) and > (right angle bracket). When used in the phrase “enclosed in angle brackets,” the symbol < immediately precedes the object to be enclosed, and > immediately follows it. When describing these characters in the portable character set, the names <less-than-sign> and <greater-than-sign> are used. *X/Open.*

**animate.** Make or design in such a way as to create apparently spontaneous, lifelike movement.

**animation rate.** The number of thousandths of a second that pass before the next bitmap is displayed for a button while it is animated.

**anonymous union.** A union that is declared within a structure or class and that does not have a name.

**ANSI (American National Standards Institute).** An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States. *ANSI.*

**application.** (1) The use to which an information processing system is put; for example, a payroll application, an airline reservation application, a network application. *IBM.* (2) A collection of software components used to perform specific types of user-oriented work on a computer. *IBM.*

**array.** An aggregate that consists of data objects, with identical attributes, each of which may be uniquely referenced by subscripting.

**array element.** A data element in an array.

**array implementation.** (In Collection Class Library) Implementation of an abstract data type using an array. Also called a tabular implementation.

**ASCII (American National Standard Code for Information Interchange).** The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

**Note:** IBM has defined an extension to ASCII code (characters 128-255).

**assignment expression.** An operation that stores the value of the right operand in the storage location specified by the left operand.

**audio attributes.** The standard audio attributes are: mute, volume, balance, treble, and bass.

**audio formats.** The way the audio information is stored and interpreted.

**audio track.** (1) The audio (sound) portion of the program. (2) The physical location where the

## automatic data •brackets

audio is placed beside the image. (A system with two sound tracks can have either stereo sound or two independent sound tracks.) Synonymous with sound track.

**automatic data.** Data that does not persist after a routine has finished executing. Automatic data may be automatically initialized to a certain value upon entry and reentry to a routine.

**automatic storage.** Storage that is allocated on entry to a routine or block and is freed on the subsequent return. Sometimes referred to as *stack storage* or *dynamic storage*.

**automatic storage management.** The process that automatically allocates and deallocates objects in order to use memory efficiently.

**auto-reset event.** In Windows, an event used to signal a single thread that an application has completed. See also event.

**auxiliary classes.** Classes that support other classes. Auxiliary classes in the Collection Class Library include classes for cursors, pointers, and iterators.

**AVL tree.** A balanced binary search tree that does not allow the height of two siblings to differ by more than one.

## B

**B\*-tree (B star tree).** A tree in which only the leaves contain whole elements. All other nodes contain keys.

**background color.** The color in which the background of a graphic primitive is drawn.

**backslash.** The character \. This character is named <backslash> in the portable character set.

**balance.** (1) For audio, refers to the relative strength of the left and right channels. A balance level of 0 is left channel only. A balance level of 100 is right channel only (2) A state of equilibrium, usually between treble and bass.

**base class.** A class from which other classes are derived. A base class may itself be derived from another base class. See also abstract class.

**based on.** A relationship between two classes in which one class is implemented through the other. A new class is “based on” an existing class when the existing class is used to implement it.

**binary expression.** An operation containing two operands and one operator.

**binary stream.** (1) An ordered sequence of untranslated characters. (2) A sequence of characters that corresponds on a one-to-one basis with the characters in the file. No character translation is performed on binary streams. *IBM*.

**bit field.** A member of a structure or union that contains a specified number of bits.

**bit mask.** A pattern of characters used to control the retention or elimination of portions of another pattern of characters.

**bits-per-sample.** The number of bits of audio data that is to represent each sample of each channel (right or left). This is the resolution of the audio data. CD quality needs to be 16 bits-per-sample.

**block.** (1) In programming languages, a compound statement that coincides with the scope of at least one of the declarations contained within it. A block may also specify storage allocation or segment programs for other purposes. *ISO-JTC1*. (2) A string of data elements recorded or transmitted as a unit. The elements may be characters, words, or physical records. *ISO Draft*. (3) The unit of data transmitted to and from a device. Each block contains one record, part of a record, or several records.

**boundary alignment.** The position in main storage of a fixed-length field (such as byte or doubleword) on an integral boundary for that unit of information.

For the Class Library example, a word boundary is a storage address evenly divisible by two.

**bounded collection.** A collection that has an upper limit on the number of elements it can contain.

**brackets.** The characters [ (left bracket) and ] (right bracket), also known as *square brackets*. When used in the phrase “enclosed in (square) brackets” the symbol [ immediately precedes the

## breakpoint • character class

object to be enclosed, and ] immediately follows it. When describing these characters in the portable character set, the names <left-bracket> and <right-bracket> are used. *X/Open*.

**breakpoint.** A point in a computer program where execution may be halted. A breakpoint is usually at the beginning of an instruction where halts caused by external intervention are convenient for resuming execution. *ISO Draft*.

**build.** An action that invokes the WorkFrame Build tool. The Build tool manages the project's makefile, as well as build dependencies between projects in a project hierarchy.

**build actions.** A series of actions that are invoked to build a project's target. These actions are set in the Build options window, or in MakeMake, WorkFrame's makefile creation utility.

**brightness.** The level of luminosity of the video signal. A brightness level of 0 produces a maximally white signal. A brightness level of 100 produces a maximally black signal.

**built-in.** A function that the compiler automatically puts inline instead of generating a call to the function. Synonymous with predefined. *IBM*.

## C

**C++ class library.** See *class library*.

**C++ library.** A system library that contains common C++ language subroutines for file access, memory allocation, and other functions.

**call.** To transfer control to a procedure, program, routine, or subroutine. *IBM*.

**caller.** A routine that calls another routine.

**canvas.** Canvases are windows with a layout algorithm that manages child windows. The canvas classes are a set of window classes that allow you to implement dialog boxes. These dialog windows are used for showing views of objects as both pages in a notebook and as windows that gather information to run an action. The different canvases can manage the size and position of child windows, provide moveable split

bars between windows, and support the ability to scroll a window.

The canvases include the base class, ICanvas, and its four derived classes: IMultiCellCanvas, ISetCanvas, ISplitCanvas, and IViewport.

**carriage-return character.** A character that in the output stream indicates that printing should start at the beginning of the same physical line in which the carriage-return character occurred. The carriage-return is the character designated by '\r' in the C and C++ languages. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the movement to the beginning of the line. *X/Open*.

**CASE.** Computer-Aided Software Engineering.

**cast.** A notation used to express the conversion of one type to another.

**catch block.** A block associated with a try block that receives control when a C++ exception matching its argument is thrown.

**CD-XA.** Compact disc-extended architecture.

**channel mapping.** The translation of a MIDI channel number for a sending device to an appropriate channel for a receiving device.

**character.** (1) A letter, digit, or other symbol that is used as part of the organization, control, or representation of data. A character is often in the form of a spatial arrangement of adjacent or connected strokes. *ANSI*. (2) A sequence of one or more bytes representing a single graphic symbol or control code. This term corresponds to the ISO C standard term *multibyte character* (multi-byte character), where a single-byte character is a special case of the multi-byte character. Unlike the usage in the ISO C standard, *character* here has no necessary relationship with storage space, and *byte* is used when storage space is discussed. *X/Open. ISO.1*.

**character array.** An array of type char. *X/Open*.

**character class.** A named set of characters sharing an attribute associated with the name of the class. The classes and the characters that they contain are dependent on the value of the LC\_CTYPE category in the current locale. *X/Open*.



## character constant • code point

**character constant.** (1) A constant with a character value. *IBM.* (2) A string of any of the characters that can be represented, usually enclosed in apostrophes. *IBM.* (3) In some languages, a character enclosed in apostrophes. *IBM.*

**character set.** (1) A finite set of different characters that is complete for a given purpose; for example, the character set in ISO Standard 646, 7-bit Coded Character Set for Information Processing Interchange. *ISO Draft.* (2) All the valid characters for a programming language or for a computer system. *IBM.* (3) A group of characters used for a specific reason; for example, the set of characters a printer can print. *IBM.* (4) See also *portable character set.*

**character string.** A contiguous sequence of characters terminated by and including the first null byte. *X/Open.*

**child.** A node that is subordinate to another node in a tree structure. Only the root node of a tree is not a child.

**child class.** See *derived class.*

**child window.** A window derived from another window and drawn relative to it.

**circular slider control.** A 360-degree knob-like control that simulates the buttons on a TV, a stereo, or video components. By rotating the slider arm, the user can set, display, or modify a value, such as the balance, bass, volume, or treble.

**class.** (1) A group of objects that share a common definition and that therefore share common properties, operations, and behavior. (2) A C++ aggregate that may contain functions, types, and user-defined operators in addition to data. Classes can be defined hierarchically, allowing one class to be an expansion of another, and classes can restrict access to their members.

**class hierarchy.** A tree-like structure showing relationships among classes. It places one abstract class at the top (a base class) and one or more layers of derived classes below it.

**class identifier (CLSID).** In Windows 95, the globally unique identifier for an object. The Win32 system registration database uses the

CLSID to distinguish all OLE objects available on a system. A CLSID is a 16-bit value that contains 32 hexadecimal digits.

Also commonly referred to as a globally unique identifier (GUID).

**class key.** Any of the three C++ keywords: **class**, **struct**, or **union**.

**class library.** A collection of classes.

**class scope.** The scope of class member names.

**class template.** A blueprint describing how a set of related classes can be constructed.

**class name.** A unique identifier of a class type that becomes a reserved word within its scope.

**class tag.** See *class name.*

**client area window.** An intermediate window between an *IFrameWindow* and its controls and other child windows.

**C library.** A system library that contains common C language subroutines for file access, string operators, character operations, memory allocation, and other functions. *IBM.*

**client program.** A program that uses a class. The program is said to be a client of the class.

**CLSID.** Class identifier.

**coded character set.** (1) A set of graphic characters and their code point assignments. The set may contain fewer characters than the total number of possible characters: some code points may be unassigned. *IBM.* (2) A coded set whose elements are single characters; for example, all characters of an alphabet. *ISO Draft.* (3) Loosely, a code. *ANSI.*

**code page.** (1) An assignment of graphic characters and control function meanings to all code points; for example, assignment of characters and meanings to 256 code points for an 8-bit code, or assignment of characters and meanings to 128 code points for a 7-bit code. (2) A particular assignment of hexadecimal identifiers to graphic characters.

**code point.** (1) A 1-byte code representing one of 256 potential characters. (2) An identifier in

## collating element •Complex Mathematics library

an alert description that represents a short unit of text. The code point is replaced with the text by an alert display program.

**collating element.** The smallest entity used to determine the logical ordering of character or wide-character strings. A collating element consists of either a single character, or two or more characters collating as a single entity. The value of the LC\_COLLATE category in the current locale determines the current set of collating elements. *X/Open*.

**collating sequence.** (1) A specified arrangement used in sequencing. *ISO-JTC1. ANSI*. (2) An ordering assigned to a set of items, such that any two sets in that assigned order can be collated. *ANSI*. (3) The relative ordering of collating elements as determined by the setting of the LC\_COLLATE category in the current locale. The character order, as defined for the LC\_COLLATE category in the current locale, defines the relative order of all collating elements, such that each element occupies a unique position in the order. This is the order used in ranges of characters and collating elements in regular expressions and pattern matching. In addition, the definition of the collating weights of characters and collating elements uses collating elements to represent their respective positions within the collation sequence.

**collation.** The logical ordering of character or wide-character strings according to defined precedence rules. These rules identify a collation sequence between the collating elements, and such additional rules that can be used to order strings consisting of multiple collating elements. *X/Open*.

**collection.** (1) In a general sense, an implementation of an abstract data type for storing elements. (2) An abstract class without any ordering, element properties, or key properties. All abstract Collection Classes are derived from Collection.

**Collection Class Library.** A set of classes that provide basic functions for collections, and can be used as base classes.

**Collection Classes.** A set of classes that implement abstract data types for storing elements.

**color palette.** A set of all the colors that can be used in a displayed image.

**COM.** Component Object Model.

**comma expression.** An expression that contains two operands separated by a comma. Although the compiler evaluates both operands, the value of the expression is the value of the right operand. If the left operand produces a value, the compiler discards this value. Typically, the left operand of a comma expression is used to produce side-effects.

**command.** A request to perform an operation or run a program. When parameters, arguments, flags, or other operands are associated with a command, the resulting character string is a single command.

**common controls.** In Windows, a DLL that includes the following: a header control (a window for displaying multiple columns of data), list view (a way to display objects as icons with labels), progress bar, property sheet, status bar, tool bar, track bar (slider control), tree view (an outline-type list), and an up-down control (spin control).

**compact disc-extended architecture (CD-EX).** A storage format that accommodates interleaved storage of audio, video, and standard file system data.

**compact disc-read-only memory (CD-ROM).** High-capacity, read-only memory in the form of an optically read compact disc.

**compilation unit.** (1) A portion of a computer program sufficiently complete to be compiled correctly. *IBM*. (2) A single compiled file and all its associated include files. (3) An independently compilable sequence of high-level language statements. Each high-level language product has different rules for what makes up a compilation unit.

**complete class name.** The complete qualification of a nested class name, including all enclosing class names.

**Complex Mathematics library.** A C++ class library that provides the facilities to manipulate complex numbers and perform standard mathematical operations on them.

## compound document •conversion

**compound document.** A means for integrating arbitrary or unstructured data from different sources into one centralized location.

**Compound Document Framework.** A starting point for creating a server or container document component that is OLE-enabled. The framework stores compound documents using the OLE-structured storage specification (docfiles).

**Compound Object Model (COM).** The underlying model for all OLE services. It consists of a variety of APIs and object interfaces that allow container components to communicate and interact with one another.

**composite.** The combination of two or more film, video, or electronic images into a single frame or display.

**Computer-Aided Software Engineering (CASE).** A set of tools or programs to help develop complex applications. *IBM*.

**computer-controlled device.** An external video source device with frame-stepping capability, usually a videodisc player, whose output can be controlled by the multimedia subsystem.

**concrete class.** A class that implements an abstract data type but does not allow polymorphism.

**condition.** (1) A relational expression that can be evaluated to a value of either true or false. *IBM*. (2) An exception that has been enabled, or recognized, by the *Language Environment* and thus is eligible to activate user and language condition handlers. Any alteration to the normal programmed flow of an application. Conditions can be detected by the hardware/operating system and result in an interrupt. They can also be detected by language-specific generated code or language library code.

**conditional expression.** A compound expression that contains a condition (the first expression), an expression to be evaluated if the condition has a nonzero value (the second expression), and an expression to be evaluated if the condition has the value zero (the third expression).

**const.** (1) An attribute of a data object that declares that the object cannot be changed. (2) An attribute of a function that declares that

the function will not modify data members of its class.

**constant.** (1) In programming languages, a language object that takes only one specific value. *ISO-JTC1*. (2) A data item with a value that does not change. *IBM*.

**constant expression.** An expression having a value that can be determined during compilation and that does not change during the running of the program. *IBM*.

**constructor.** A special class member function that has the same name as the class and is used to construct and, possibly, initialize objects of its class type. A return type is not specified.

**container.** An object that holds other objects.*IBM*. Containers built with the Compound Document Framework are also servers and can therefore be embedded inside other containers. A container can hold zero or more embedded components.

**containment function.** A function that determines whether a collection contains a given element.

**control.** A graphic object that represents operations or properties of other objects. See also tree control.

**control character.** (1) A character whose occurrence in a particular context specifies a control function. *ISO Draft*. (2) Synonymous with nonprinting character. *IBM*. (3) A character, other than a graphic character, that affects the recording, processing, transmission, or interpretation of text. *X/Open*.

**conversion.** (1) In programming languages, the transformation between values that represent the same data item but belong to different data types. Information may be lost because of conversion since accuracy of data representation varies among different data types. *ISO-JTC1*. (2) The process of changing from one method of data processing to another or from one data processing system to another. *IBM*. (3) The process of changing from one form of representation to another; for example to change from decimal representation to binary representation. *IBM*. (4) A change in the type of a value. For example, when you add values having different

## conversion function • default operation class

data types, the compiler converts both values to a common form before adding the values.

**conversion function.** A member function that specifies a conversion from its class type to another type.

**copy constructor.** A constructor used to make a copy of an object from another object of the same type.

**critical section.** (1) Code that must be executed by one thread while all other threads in the process are suspended. (2) In Windows, a synchronization object. A critical section is not a kernel object; that is, it is not managed by the low-level components of the operating system and is not manipulated using handles. (3) In Windows, a small section of code that requires exclusive access to some shared data before the code can execute. Critical threads synchronize threads only within a single process, and they allow only one thread at a time to gain access to a region of data.

See also mutex, semaphore, and event. Contrast with kernel object.

**cursor.** A reference to an element at a specific position in a data structure.

**cursored emphasis.** When the selection cursor is on a choice, that choice has cursored emphasis.

**cursor iteration.** The process of repeatedly moving the cursor to the next element in a collection until some condition is satisfied.

## D

**daemon.** A program that runs unattended to perform a service for other programs.

**data abstraction.** A data type with a private representation and a public set of operations. The C++ language uses the concept of classes to implement data abstraction.

**data member.** The smallest possible piece of complete data. Elements are composed of data members.

**data structure.** The internal data representation of an implementation.

**data type.** The properties and internal representation that characterize data.

**DBCS (Double-Byte Character Set).** See double-byte character set.

**decimal constant.** (1) A numerical data type used in standard arithmetic operations. (2) A number containing any of the digits 0 through 9. *IBM.*

**deck.** A line of child windows in a set canvas that is direction-independent. A horizontal deck is equivalent to a row and a vertical deck is equivalent to a column.

**declaration.** Introduces a name to a program and specifies how the name is to be interpreted.

**declare.** To specify the interpretation that C++ gives to each identifier.

**default argument.** An argument that is declared with a default value in a function prototype or declaration. If a call to the function omits this argument, the default value is used. Arguments with default values must be the trailing arguments in a function prototype argument list.

**default class.** A class with preprogrammed definitions that can be used for simple implementations.

**default constructor.** A constructor that takes no arguments, or a constructor for which all the arguments have default values.

**default implementation.** One of several possible implementation variants offered as the default for a specific abstract data type.

**default initialization.** The initial value assigned to a data object by the compiler if no initial value is specified by the programmer. **extern** and **static** variables receive a default initialization of zero, while the default initial value for **auto** and **register** variables is undefined.

**default operation class.** A class with preprogrammed definitions for all required element and key operations for a particular implementation.

## define directive • double-precision

**define directive.** A preprocessor statement that directs the preprocessor to replace an identifier or macro invocation with special code.

**definition.** (1) A data description that reserves storage and may provide an initial value. (2) A declaration that allocates storage, and may initialize a data object or specify the body of a function.

**degree.** The number of children of a node.

**delete.** (1) A C++ keyword that identifies a free-storage deallocation operator. (2) A C++ operator used to destroy objects created by operator new.

**demangling.** The conversion of mangled names back to their original source code names. During C++ compilation, identifiers such as function and static class member names are mangled (encoded) with type and scoping information to ensure type-safe linkage. These mangled names appear in the object file and the final executable file. Demangling (decoding) converts these names back to their original names to make program debugging easier. See also *mangling*.

**denormal.** Pertaining to a number with a value so close to 0 that its exponent cannot be represented normally. The exponent can be represented in a special way at the possible cost of a loss of significance.

**deque.** A queue that can have elements added and removed at both ends. A double-ended queue.

**dequeue.** An operation that removes the first element of a queue.

**derivation.** (1) The creation of a new or derived class from an existing base class. (2) The relationship between a class and the classes above or below it in a class hierarchy.

**derived class.** A class that inherits from a base class. You can add new data members and member functions to the derived class. You can manipulate a derived class object as if it were a base class object. The derived class can override virtual functions of the base class.

Synonym for *child class* and *subclass*.

**destructor.** A special member function that has the same name as its class, preceded by a tilde (~), and that “cleans up” after an object of that class, for example, by freeing storage that was allocated when the object was created. A destructor has no arguments, and no return type is specified.

**device.** A computer peripheral or an object that appears to the application as such. *X/Open. ISO.1.*

**difference.** Given two sets A and B, the difference (A-B) is the set of all elements contained in A but not in B.

**diluted array.** An array in which elements are deleted by being flagged as deleted, rather than by actually removing them from the array and shifting later elements to the left.

**diluted sequence.** A sequence implemented using a diluted array.

**directory.** A type of file containing the names and controlling information for other files or other directories. *IBM.*

**display.** To direct the output to the user's terminal. If the output is not directed to the terminal, the results are undefined. *X/Open.*

**document component.** The basic unit of data exchange in the Compound Document Framework.

See also *component*.

**double-byte character set (DBCS).** A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets.

Because each character requires 2 bytes, you need hardware and supporting software that are DBCS-enabled to enter, display, and print DBCS characters.

**double-precision.** Pertaining to the use of two computer words to represent a number in accordance with the required precision. *ISO-JTC1. ANSI.*

## doubleword • enumerator

**doubleword.** A contiguous sequence of bits or characters that comprises two computer words and can be addressed as a unit. For the C Set++ for AIX compiler, a doubleword is 32 bits (4 bytes).

**drag after.** A target enter event that occurs in a container where its `orderedTargetEmphasis` or `mixedTargetEmphasis` attribute is set and the current view is name, text, or details.

**drag item.** A “proxy” for the object being manipulated.

**drag over.** A target enter event that occurs in a container where its `orderedTargetEmphasis` attribute is not set and the current view is icon or tree view.

**drop offset.** The location where the next container object that is dropped will be positioned (if the target operation’s drop style is not `IDM::dropPosition`). The position is based upon the last object that was dropped as an offset of that object relative to the drop style.

**dump.** To copy data in a readable format from main or auxiliary storage onto an external medium such as tape, diskette, or printer. *IBM.*

**dynamic.** Pertaining to an operation that occurs at the time it is needed rather than at a predetermined or fixed time. *IBM.*

**dynamic binding.** Resolution of a call to a virtual member function at run time.

**dynamic casting.** An intelligent mechanism for obtaining the correct pointer to a base class.

**dynamic link library (DLL).** A file containing executable code and data bound to a program at load time or run time. The code and data in a dynamic link library can be shared by several applications simultaneously.

**dynamic storage.** Synonym for *automatic storage*.

## E

**EBCDIC (extended binary-coded decimal interchange code).** A coded character set of 256 8-bit characters.

**element.** The component of an array, subrange, enumeration, or set.

**element equality.** A relation that determines whether two elements are equal.

**element function.** A function, called by a member function, that accesses the elements of a class.

**embedded component.** Data that was created by another application and stored in a container’s structured storage file. Unlike a linked object, an embedded component does not have its own file on disk.

**empty string.** (1) A string whose first byte is a null byte. Synonymous with null string. *X/Open.*  
(2) A character array whose first element is a null character. *ISO.1.*

**encapsulation.** The hiding of the internal representation of objects and implementation details from the client program.

**enqueue.** An operation that adds an element as the last element to a queue.

**entry point.** In assembler language, the address or label of the first instruction that is executed when a routine is entered for execution.

**enumeration constant.** An identifier that is defined in an enumeration and that has an associated constant integer value. You can use an enumeration constant anywhere an integer constant is allowed.

**enumeration data type.** A type that represents integers and a set of enumeration constants. Each enumeration constant has an associated integer value.

**enumeration tag.** The identifier that names an enumeration data type.

**enumerator.** In the C and C++ language, an enumeration constant and its associated value. *IBM.*

## environment variable • executable file

**environment variable.** Any of a number of variables that describe the way an operating system is going to run or the devices it is going to recognize. *IBM*.

**equality collection.** (1) An abstract class with the property of element equality. (2) In general, any collection that has element equality.

**equality key collection.** An abstract class with the properties of element equality and key equality.

**equality key sorted collection.** An abstract class with the properties of element equality, key equality, and sorted elements.

**equality sequence.** A sequentially ordered flat collection with element equality.

**equality sorted collection.** An abstract class with the properties of element equality and sorted elements.

**equivalence class.** (1) A grouping of characters that are considered equal for the purpose of collation; for example, many languages place an uppercase character in the same equivalence class as its lowercase form, but some languages distinguish between accented and unaccented character forms for the purpose of collation. *IBM*. (2) A set of collating elements with the same primary collation weight.

Elements in an equivalence class are typically elements that naturally group together, such as all accented letters based on the same base letter.

The collation order of elements within an equivalence class is determined by the weights assigned on any subsequent levels after the primary weight. *X/Open*.

**escape sequence.** (1) A representation of a character. An escape sequence contains the \ symbol followed by one of the characters: a, b, f, n, r, t, v, ' , " , x, \, or followed by one or more octal or hexadecimal digits. (2) A sequence of characters that represent, for example, nonprinting characters, or the exact code point value to be used to represent variant and nonvariant characters regardless of code page. (3) In the C and C++ language, an escape character followed by one or more characters. The escape character indicates that a different code, or a different

coded character set, is used to interpret the characters that follow. Any member of the character set used at runtime can be represented using an escape sequence. (4) A character that is preceded by a backslash character and is interpreted to have a special meaning to the operating system. (5) A sequence sent to a terminal to perform actions such as moving the cursor, changing from normal to reverse video, and clearing the screen. Synonymous with multibyte control. *IBM*.

**event.** (1) Any user action (such as a mouse click) or system activity (such as screen updating) that provokes a response from the application. (2) In Windows, a synchronization kernel object used to signal that an operation has completed. See also kernel object. Compare to critical section, mutex, semaphore, manual-reset event, and auto-reset event.

**exception.** (1) A user or system error detected by the system and passed to an operating system or user exception handler. (2) For C++, any user, logic, or system error detected by a function that does not itself deal with the error but passes the error on to a handling routine (also called “throwing the exception”).

**exception handler.** (1) A function that is invoked when an exception is detected, and that either corrects the problem and returns execution to the program, or terminates the program. (2) In C++, a catch block that catches a C++ exception when it is thrown from a function in a try block.

**exception handling.** A type of error handling that allows control and information to be passed to an exception handler when an exception occurs. In C++, try, catch, and throw expressions are the constructs used to implement C++ exception handling.

**executable file.** A regular file acceptable as a new process image file by the equivalent of the *exec* family of functions, and thus usable as one form of a utility. The standard utilities described as compilers can produce executable files, but other unspecified methods of producing executable files may also be provided. The internal format of an executable file is unspecified, but a conforming application cannot assume an executable file is a text file. *X/Open*.

## extension •function scope

**extension.** (1) An element or function not included in the standard language. (2) File name extension.

**external data definition.** A definition appearing outside a function. The defined object is accessible to all functions that follow the definition and are located within the same source file as the definition.

## F

**file descriptor.** A small positive integer that the system uses instead of the file name to identify an open file.

**file scope.** A name declared outside all blocks and classes has file scope and can be used after the point of declaration in a source file.

**file-scoped action.** Distinguished from a project-scoped action in that it is invoked on files. Only file-scoped actions can participate in a project build.

**filter.** (1) A command whose operation consists of reading data from standard input or a list of input files and writing data to standard output. Typically, its function is to perform some transformation on the data stream. (2) In WorkFrame, the value of a type. The filter of a type can be expressed as a file mask; a regular expression; a logical-OR, logical-AND, or logical-NOT of a list of types; or a filter determined by a PAM.

**first element.** The element visited first in an iteration over a collection. Each collection has its own definition for first element. For example, the first element of a sorted set is the element with the smallest value.

**flat collection.** A collection that has no hierarchical structure.

**folder.** A directory.

**font.** A particular size and style of typeface that contains definitions of character sets, marker sets, and pattern sets.

**for statement.** A looping statement that contains the word *for* followed by a list of expressions enclosed in parentheses (the condition) and a

statement (the action). Each expression in the parenthesized list is separated by a semicolon. You can omit any of the expressions, but you cannot omit the semicolons.

**free store.** Dynamically allocated memory. New is used to allocate free store and delete to deallocate it.

**friend class.** A class in which all the member functions are granted access to the private and protected members of another class. It is named in the declaration of the other class with the prefix friend.

**friend function.** A function that is granted access to the private and protected parts of a class. It is named in the declaration of the class with the prefix friend.

**function.** A named group of statements that can be called and evaluated and can return a value to the calling statement. *IBM.* See also member function.

**function call.** An expression that moves the path of execution from the current function to a specified function and evaluates to the return value provided by the called function. A function call contains the name of the function to which control moves and a parenthesized list of values. *IBM.*

**function declarator.** The part of a function definition that names the function, provides additional information about the return value of the function, and lists the function parameters.

**function definition.** The complete description of a function. A function definition contains an optional storage class specifier, an optional type specifier, a function declarator, optional parameter declarations, and a block statement (the function body).

**function prototype.** A function declaration that provides type information for each parameter. It is the first line of the function (header) followed by a ; (semicolon). The declaration is required by the compiler at the time that the function is declared, so that the compiler can check the type.

**function scope.** The capacity to be used anywhere in a function. Labels that are declared in a function have function scope.



## function template • identifier

**function template.** Provides a blueprint describing how a set of related individual functions can be constructed.

## G

**GDI.** Graphics device interface.

**generic class.** See *class templates*.

**global.** Pertaining to information available to more than one program or subroutine. *IBM*.

**global scope.** See *file scope*.

**global variable.** A symbol defined in one program module that is used in other independently compiled program modules.

**graphic attributes.** Attributes that apply to graphic primitives. Examples are color, line type, and shading-pattern definition.

**graphic primitive.** A single item of drawn graphics, such as a line, arc, or graphics text string.

**graphical user interface (GUI).** Type of computer interface consisting of a visual metaphor of a real-world scene, often of a desktop.

**graphics.** A picture defined in terms of graphic primitives and graphic attributes.

**Graphics device interface object.** An object such as a brush, pen, or bitmap. All graphics device interface (GDI) objects are owned by the process that also owns the thread.

Compare to user object and kernel object.

**GUI.** Graphical user interface.

**GUID.** The globally unique identifier for an object. See also *class identifier*.

## H

**halftone.** The reproduction of continuous-tone artwork, such as a photograph, by converting the image into dots of various sizes.

**hash function.** A function that determines which category, or bucket, to put an element in. A hash function is needed when implementing a hash table.

**hash table.** A data structure that divides all elements into (preferably) equal-sized categories, or buckets, to allow quick access to the elements. The hash function determines which bucket an element belongs in.

**header file.** A text file that is used by a group of functions, programs, or users. It can contain system-defined control information or user data and generally consists of declarations. Synonymous with include file.

**heap.** An unordered flat collection that allows duplicate elements.

**heap storage.** An area of storage used for allocation of storage whose lifetime is not related to the execution of the current routine. The heap consists of the initial heap segment and zero or more increments.

**height of a tree.** The length of the longest path from the root to a leaf.

**hexadecimal constant.** A constant, usually starting with special characters, that contains only hexadecimal digits. Three examples for the hexadecimal constant with value 0 would be `'\x00'`, `'0x0'`, or `'0X00'`.

**hit testing.** The means of identifying which graphic object the mouse is pointing to.

## I

**identifier.** (1) One or more characters used to identify or name a data element and possibly to indicate certain properties of that data element.

*ANSI.* (2) In programming languages, a token that names a data object such as a variable, an array, a record, a subprogram, or a function.

*ANSI.* (3) A sequence of letters, digits, and

## if statement • instantiate

underscores used to identify a data object or function. *IBM*.

**if statement.** A conditional statement that contains the keyword *if*, followed by an expression in parentheses (the condition), a statement (the action), and an optional *else* clause (the alternative action). *IBM*.

**implementation class.** A class that implements a concrete class. Implementation classes are never used directly.

**include directive.** A preprocessor directive that causes the preprocessor to replace the statement with the contents of a specified file.

**include file.** A text file that contains declarations used by a group of functions, programs, or users. Synonymous with header file. *IBM*.

**include statement.** In the C and C++ languages, a preprocessor statement that causes the preprocessor to replace the statement with the contents of a specified file. *IBM*.

**incomplete class declaration.** A class declaration that does not define any members of a class. Typically, you use an incomplete class declaration as a forward declaration.

**incomplete type.** A type that has no value or meaning when it is first declared. There are three incomplete types: void, arrays of unknown size and structures and unions of unspecified content. A void type can never be completed. Arrays of unknown size and structures or unions of unspecified content can be completed in further declarations.

**indirection.** (1) A mechanism for connecting objects by storing, in one object, a reference to another object. (2) In the C and C++ languages, the application of the unary operator *\** to a pointer to access the object the pointer points to.

**inheritance.** (1) An object-oriented programming technique that allows you to use existing classes as bases for creating other classes. (2) A mechanism by which a derived class can use the attributes, relationships, and member functions defined in more abstract classes related to it (its base classes). See also multiple inheritance.

**initializer.** An expression used to initialize objects. In the C++ language, there are three types of initializers:

1. An expression followed by an assignment operator is used to initialize fundamental data type objects or class objects that have copy constructors.
2. An expression enclosed in braces ( { } ) is used to initialize aggregates.
3. A parenthesized expression list is used to initialize base classes and members using constructors.

**inlined function.** A function call that the compiler replaces with the actual code for the function. You can direct the compiler to inline a function with the *inline* keyword.

**in-place activation.** A container and a server merging elements of their user interface, such as small child windows, toolbars, and menus, into the container's window space. This allows the end-user access to the server's controls from within the container, thereby providing a more document-centric approach to working. Simple activation of a server, is a more application-centric approach to application interaction. Also called *in situ editing*.

**input stream.** A sequence of control statements and data submitted to a system from an input unit.

**in situ editing.** See *in-place activation*.

**instance (of a class).** An object that is a member of that class. An object created according to the definition of that class.

**instance number.** A number that the operating system uses to keep track of all of the instances of the same type of device. For example, the amplifier-mixer device name is AMPMIX plus a 2-digit instance number. If a program creates two amplifier-mixer objects, the device names could be AMPMIX01 and AMPMIX02.

**instantiate.** To create or generate a particular instance or object of a data type. For example, an instance *box1* of class *box* could be instantiated with the declaration:

```
box box1;
```

## instruction •key sorted collection

**instruction.** A program statement that specifies an operation to be performed by the computer, along with the values or locations of operands. This statement represents the programmer's request to the processor to perform a specific operation.

**instruction scheduling.** An optimization technique that reorders instructions in code to minimize execution time.

**integer constant.** A decimal, octal, or hexadecimal constant.

**integral object.** A character object, an object having an enumeration type, an object having variations of the type `int`, or an object that is a bit field.

**interactive graphics.** Graphics that a user at a terminal can move or manipulate.

**interactive video.** The process of combining video and computer technology so that the user's actions, choices, and decisions affect the way in which the program unfolds.

**internationalization.** The capability of a computer program to adapt to the requirements of different native languages, local customs, and coded character sets. *X/Open*.

**interrupt.** A temporary suspension of a process caused by an external event, performed in such a way that the process can be resumed.

**intersection.** Given collections A and B, the set of elements that is contained in both A and B.

**intrinsic function.** A function supplied by a program as opposed to a function supplied by the compiler.

**inverted colors.** Opposite colors in the light spectrum.

**I/O Stream Library.** A class library that provides the facilities to deal with many varieties of input and output.

**iteration.** The process of repeatedly applying a function to a series of elements in a collection until some condition is satisfied.

**iteration order.** The order in which elements are accessed when iterating over a collection. In

ordered collections, the element at position 1 will be accessed first, then the element at position 2, and so on. In sorted collections, the elements are accessed according to the ordering relation provided for the element type. In collections that are not ordered the elements are accessed in an arbitrary order. Each element is accessed exactly once.

**iterator class.** A class that provides iteration functions.

## K

**kernel.** The core of an operating system, usually responsible for basic I/O and process execution.

**kernel object.** In Windows, an object used by the system and your applications to manage numerous resources, such as processes, threads, and files.

Compare to graphics device interface object and user object.

**key access.** A property that allows elements to be accessed by matching keys.

**key bag.** An unordered flat collection that uses keys and can contain duplicate elements.

**key collection.** (1) An abstract class that has the property of key access. (2) In general, any collection that uses keys.

**key equality.** A relation that determines whether two keys are equal.

**key function.** A function with the name `key()` which, when used on a flat collection, returns a reference to the key of an element.

**key-type function.** Any of several functions of an element type, that are used by the Collection Class Library member functions to manipulate the keys of a class.

**key set.** An unordered flat collection that uses keys and does not allow duplicate elements.

**key sorted bag.** A sorted flat collection that uses keys and allows duplicate elements.

**key sorted collection.** An abstract class with the properties of key equality and sorted elements.

## key sorted set • lvalue

**key sorted set.** A sorted flat collection that uses keys and does not allow duplicate elements.

**keyword.** (1) A predefined word reserved for the C or C++ language that you cannot use as an identifier. (2) A symbol that identifies a parameter.

## L

**label.** An identifier within or attached to a set of data elements.

*ISO Draft.*

**last element.** The element accessed last in an iteration over a collection. Each collection has its own definition for last element. For example, the last element of a sorted set is the element with the largest value.

**latched.** The state of a button. A button in its latched state is held in its pressed position until the user clicks on it to release (unlatch) it.

**late binding.** See *dynamic binding*.

**leaves.** In a tree, nodes without children. Synonymous with terminals.

**lexically.** Relating to the left-to-right order of units.

**library.** (1) A collection of functions, function calls, subroutines, or other data. (2) A set of object modules that can be specified in a link command.

**link.** To interconnect items of data or portions of one or more computer programs; for example, linking of object programs by a linkage editor to produce an executable file.

**linkage editor.** Synonym for linker.

**linked component.** A component with data stored outside the compound document. The compound document includes a moniker that names that other location.

**linked implementation.** An implementation in which each element contains a reference to the next element in the collection. Pointer chains are used to access elements in linked

implementations. Linked implementations are also called linked list implementations.

**linked sequence.** A sequence that uses a linked implementation.

**linker.** A program that resolves cross-references between separately compiled object modules and then assigns final addresses to create a single executable program.

**literal.** (1) In programming languages, a lexical unit that directly represents a value; for example, 14 represents the integer fourteen, "APRIL" represents the string of characters APRIL, 3.0005E2 represents the number 300.05. *ISO-JTC1.* (2) A symbol or a quantity in a source program that is itself data, rather than a reference to data. *IBM.* (3) A character string whose value is given by the characters themselves; for example, the numeric literal 7 has the value 7, and the character literal CHARACTERS has the value CHARACTERS. *IBM.*

**loader.** A routine, commonly a computer program, that reads data into main storage. *ANSI.*

**load module.** All or part of a computer program in a form suitable for loading into main storage for execution. A load module is usually the output of a linkage editor. *ISO Draft.*

**local.** (1) In programming languages, pertaining to the relationship between a language object and a block such that the language object has a scope contained in that block. *ISO-JTC1.* (2) Pertaining to that which is defined and used only in one subdivision of a computer program. *ANSI.*

**locale.** The definition of the subset of a user's environment that depends on language and cultural conventions.

**localization.** The process of establishing information within a computer system specific to the operation of particular native languages, local customs, and coded character sets. *X/Open.*

**local scope.** A name declared in a block, which can only be used in that block.

**lvalue.** An expression that represents an object that can be both examined and altered.

## M

**macro.** An identifier followed by arguments (may be a parenthesized list of arguments) that the preprocessor replaces with the replacement code located in a preprocessor `#define` directive.

**main function.** An external function with the identifier `main` that is the first user function—aside from exit routines and C++ static object constructors—to get control when program execution begins. Each C and C++ program must have exactly one function named `main`.

**make.** An action in which a project's target is built from a makefile by a make utility.

**makefile.** A text file containing a list of your application's parts. The make utility uses makefiles to maintain application parts and dependencies.

**MakeMake.** WorkFrame's makefile generation utility.

**mangling.** The encoding during compilation of identifiers such as function and variable names to include type and scope information. The prelinker uses these mangled names to ensure type-safe linkage. See also *demangling*.

**manipulator.** A value that can be inserted into streams or extracted from streams to affect or query the behavior of the stream.

**manual-reset event.** In Windows, an event used to signal several threads simultaneously that an operation has completed.

See also *event*.

**map file.** A listing file that can be created during the prelink or link step and that contains information on the size and mapping of segments and symbols.

**mask.** A pattern of bits or characters that controls the keeping, deleting, or testing of portions of another pattern of bits or characters. *ISO-ITCI. ANSI.*

**MBCS.** See multibyte character set.

**MDI.** See multiple document interface.

**member.** Data, functions, or types contained in classes, structures, or unions.

**member function.** (1) In C++, an operator or function that is declared as a member of a class. A member function has access to the private and protected data members and member functions of objects of its class. Member functions are also called methods. (2) A function that performs operations on a class.

**message.** A request from one object that the receiving object implement a method. Because data is encapsulated and not directly accessible, a message is the only way to send data from one object to another. Each message specifies the name of the receiving object, the method to be implemented, and any parameters the method needs for implementation.

**method.** Synonym for member function.

**MIDI.** Musical Instrument Digital Interface. A standard used in the music industry for interfacing digital musical instruments.

**migrate.** To move to a changed operating environment, usually to a new release or version of a system. *IBM.*

**mix.** (1) An attribute that determines how the foreground of a graphic primitive is combined with the existing color of graphics output. Also known as foreground mix. Contrast with background mix. (2) The combination of audio or video sources during postproduction.

**mixer.** A device used to simultaneously combine and blend several inputs into one or two outputs.

**mode.** A collection of attributes that specifies a file's type and its access permissions. *X/Open. ISO.1.*

**model.** In Compound Document Framework, the data portion of a document component. The model and the view comprise the two pieces of a document component. Compound Document Framework provides an *IModel* base class from which other model classes can be derived.

**module.** A program unit that usually performs a particular function or related functions, and that is distinct and identifiable with respect to compiling, combining with other units, and loading.

## module definition file •National Television Standard Committee (NTSC)

**module definition file.** A file used by the linker that contains module statements that define general attributes of the executable being linked, segment attributes, and imported or exported functions and data.

**Monitor.** A window that displays output from monitored actions. The Monitor window is attached to the project container.

**monitored action.** An action that has been set to run in the **Monitor** window.

**motion video.** Video that displays real motion.

**mount.** (1) To place a data medium in a position to operate. (2) To make recording media accessible. (3)

**multibyte character.** A mixture of single-byte characters from a single-byte character set and double-byte characters from a double-byte character set.

**multibyte character set (MBCS).** A character set whose characters consist of more than 1 byte. Used in languages such as Japanese, Chinese, and Korean, where the 256 possible values of a single-byte character set are not sufficient to represent all possible characters.

**multicharacter collating element.** A sequence of two or more characters that collate as an entity. For example, in some coded character sets, an accented character is represented by a non-spacing accent, followed by the letter. Other examples are the Spanish elements *ch* and *ll*. *X/Open*.

**multimedia.** Computer-controlled presentations combining any of the following: text, graphics, animation, full-motion images, still video images, and sound.

**multiple document interface (MDI).** An interface that uses a primary window to contain related document windows. The parent window's title bar is displayed along with the child window's title bar. If the child window displays a document window, an icon that indicates the application data's file type appears in the child window's title bar.

**multiple inheritance.** (1) An object-oriented programming technique implemented in C++

through derivation, in which the derived class inherits members from more than one base class. (2) The structuring of inheritance relationships among classes so a derived class can use the attributes, relationships, and functions used by more than one base class.

See also inheritance and class lattice.

**multitasking.** (1) A mode of operation that allows concurrent performance or interleaved execution of more than one task or program. (2) A process that allows a computer or operating system to run multiple applications or tasks concurrently by dividing the processor's time between them rapidly.

See also preemptive multitasking. Contrast with nonpreemptive multitasking.

**multithread.** Pertaining to concurrent operation of more than one path of execution within a computer.

**multithreading.** A process that allows a multitasking operating system to multitask subportions (threads) of an application smoothly.

**mutex.** (1) In Windows, a flag that prevents threads from interacting with the 16-bit kernel when another thread is executing code there. See also nonreentrant. (2) A synchronization kernel object that synchronizes data access across multiple processes. A mutex object is either signaled or nonsignaled and is owned by a thread. See also critical section, semaphore, event, signaled, and nonsignaled.

## N

**name.** In the C++ language, a name is commonly referred to as an identifier. However, syntactically, a name can be an identifier, operator function name, conversion function name, destructor name, or qualified name.

**n-ary tree.** A tree that has an upper limit, *n*, imposed on the number of children allowed for a node.

**National Television Standard Committee (NTSC).** (1) A committee that sets the standard for color television broadcasting and video in the United States (currently in use also in Japan).

## native • null string

(2) The standard set by the NTSC committee (the NTSC standard).

**native.** The rendering mechanism and format (RMF) that best represents the object and is the best one for rendering.

For example, a native of Cincinnati understands the streets in the area better than someone who has just moved there. Therefore, a Cincinnati native can get from point A to point B quicker than a newcomer. Likewise, a native RMF can get the data transferred from point A to point B more efficiently than the additional RMFs. We can use additional RMFs when we cannot use the native, or optimal, approach.

**nested class.** A class defined within the scope of another class.

**nested project.** A project that appears inside another project. Nesting expresses a dependency of the parent project on the child project's target. This dependency is managed by WorkFrame's Build utility.

**new.** (1) A C++ keyword identifying a free storage allocation operator. (2) A C++ operator used to create class objects.

**new-line character.** A control character that causes the print or display position to move to the first position on the next line. This control character is represented by `\n` in the C language.

**node.** In a tree structure, a point at which subordinate items of data originate.

**nonmember function.** A function that occurs outside a class.

**nonpreemptive multitasking.** A type of multitasking on 16-bit Windows where one application must notify the operating system that it is finished processing before the scheduler can assign another application execution time. Also called cooperative multitasking.

Contrast with preemptive multitasking.

**nonreentrant.** The state of 16-bit code in the kernel where two threads cannot access it at the same time without risking a system crash.

In Windows 95, processes are preempted and any thread is likely to be interrupted at any point in its execution.

See also `mutex`.

**nonsignaled.** In Windows, the state that an object is in when it is suspended, or asleep. For example, when a thread is created and running, its associated thread kernel object is nonsignaled. As soon as the thread terminates, its thread kernel object is signaled.

**notification area.** (1) In Visual Builder, the information or status area at the bottom of the window. (2) In Windows 95, an area to the extreme right of the taskbar. By default (on machines with a sound driver), the notification area has two objects: a loudspeaker icon and text that shows the current time.

**NT file system (NTFS).** A Windows NT disk drive file system that restores disk-based data after a system failure. NTFS can manipulate extremely large storage media and has file names up to 255 characters in length.

**NTFS.** See NT file system.

**NTSC.** National Television Standard Committee.

**NTSC format.** The specifications for color television as defined by the NTSC, which include: (a) 525 scan lines, (b) broadcast bandwidth of 4 megaHertz, (c) line frequency of 15.75 kiloHertz, (d) frame frequency of 30 frames per second, and (e) color subcarrier frequency of 3.58 megaHertz.

**NULL.** In the C and C++ languages, a pointer that does not point to a data object. *IBM*.

**null character (\0).** The ASCII or EBCDIC character with the hex value 00 (all bits turned off). It is used to represent the absence of a printed or displayed character. This character is named `<NUL>` in the portable character set.

**null pointer.** The value that is obtained by converting the number 0 into a pointer; for example, `(void *) 0`. The C and C++ languages guarantee that this value will not match that of any legitimate pointer, so it is used by many functions that return pointers to indicate an error. *X/Open*.

**null string.** (1) A string whose first byte is a null byte. Synonymous with *empty string*.

## null value •ordering relation

*X/Open*. (2) A character array whose first element is a null character. *ISO.1*.

**null value.** A parameter position for which no value is specified. *IBM*.

**number sign.** The character #, also known as *pound sign* and *hash sign*. This character is named <number-sign> in the portable character set.

## O

**object.** (1) A computer representation of something that a user can work with to perform a task. An object can appear as text or an icon. (2) A collection of data and member functions that operate on that data, which together represent a logical entity in the system. In object-oriented programming, objects are grouped into classes that share common data definitions and member functions. Each object in the class is said to be an instance of the class. (3) In Visual Builder, an instance of an object class consisting of attributes, a data structure, and operational member functions. It can represent a person, place, thing, event, or concept. Each instance has the same properties, attributes, and member functions as other instances of the object class, though it has unique values assigned to its attributes. (4) In Windows, any item that is or can be linked into another Windows application, such as a sound, graphic, piece of text, or portion of a spreadsheet. An object must be from an application that supports OLE. See object linking and embedding (OLE).

**object code.** Machine-executable instructions, usually generated by a compiler from source code written in a higher level language (such as the C++ language). For programs that must be linked, object code consists of relocatable machine code.

**object linking and embedding (OLE).** (1) An API that supports compound documents, cross-application macro control, and common object registration. OLE defines protocols for in-place editing, drag-and-drop data transfers, structured storage, custom controls, and more. (2) A data-sharing scheme that allows dissimilar applications to create single complex documents cooperatively. The documents can consist of

material that a single application could not have created on its own.

**object module.** (1) All or part of an object program sufficiently complete for linking. Assemblers and compilers usually produce object modules. *ISO Draft*. (2) A set of instructions in machine language produced by a compiler from a source program. *IBM*.

**object-oriented programming.** A programming approach based on the concepts of data abstraction and inheritance. Unlike procedural programming techniques, object-oriented programming concentrates on what data objects comprise the problem and how they are manipulated, not on how something is accomplished.

**octal constant.** The digit 0 (zero) followed by any digits 0 through 7.

**OLE.** See object linking and embedding.

**open file.** A file that is currently associated with a file descriptor. *X/Open*. *ISO.1*.

**operation class.** A class that defines all required element and key operations required by a specific collection implementation.

**operator function.** An overloaded operator that is either a member of a class or that takes at least one argument that is a class type or a reference to a class type. See overloading.

**operator precedence.** In programming languages, an order relation defining the sequence of the application of operators within an expression. *ISO-JTC1*.

**optical reflective disc.** An optical videodisc that is read by means of the reflection of a laser beam from the shiny surface on the disc.

**ordered collection.** (1) An abstract class that has the property of ordered elements. (2) In general, any collection that has its elements arranged so that there is always a first element, last element, next element, and previous element.

**ordering relation.** A property that determines how the elements are sorted. Ascending order is an example of an ordering relation.



**overflow.** A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

**overloading.** An object-oriented programming technique where one or more function declarations are specified for a single name in the same scope.

**owner window.** A window similar to a parent window, but it does not affect the behavior or appearance of the window. The owner coordinates the activity of a window.

## P

**pack.** To store data in a compact form in such a way that the original form can be recovered.

**pad.** To fill unused positions in a field with data, usually 0's, 1's, or blanks.

**parameter.** (1) In the C and C++ languages, an object declared as part of a function declaration or definition that acquires a value on entry to the function, or an identifier following the macro name in a function-like macro definition. *X/Open*. (2) Data passed between programs or procedures. *IBM*.

**parameter declaration.** A description of a value that a function receives. A parameter declaration determines the storage class and the data type of the value.

**parent node.** A node to which one or more other nodes are subordinate.

**parent process.** (1) The program that originates the creation of other processes by means of spawn or exec function calls. See also *child process*. (2) A process that creates other processes.

**parent project.** A project that contains other projects. Distinguished from *child project*.

**parent window.** A window that provides the child window information on how and where to draw it. The parent window also defines the relationship that the child window has with other windows in the system.

**part.** In Visual Builder, a part is a self-contained object with a standardized public

interface consisting of a set of external features that allow the part to interact with other parts. A part is implemented as a class that supports the INotifier protocol and has a part interface defined.

**path name.** (1) A string that is used to identify a file. A path name consists of, at most, {PATH\_MAX} bytes, including the terminating null character. It has an optional beginning slash, followed by zero or more file names separated by slashes. If the path name refers to a directory, it may also have one or more trailing slashes. Multiple successive slashes are considered to be the same as one slash. A path name that begins with two successive slashes may be interpreted in an implementation-dependent manner, although more than two leading slashes will be treated as a single slash. The interpretation of the path name is described in *pathname resolution. ISO.1*. (2) A file name specifying all directories leading to the file.

**pattern.** A sequence of characters used either with regular expression notation or for path name expansion, as a means of selecting various characters strings or path names, respectively. The syntaxes of the two patterns are similar, but not identical. *X/Open*.

**pause.** To temporarily halt the medium. The halted visual should remain displayed but no audio should be played.

**pel.** The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonym for pixel and picture element.

**picture element.** Synonym for pel.

**pipe.** To direct data so that the output from one process becomes the input to another process. The standard output of one command can be connected to the standard input of another with the pipe operator (|). Two commands connected in this way constitute a pipeline. *IBM*.

**pitch.** The ability to change the key or keynote of the sound. For example, in music, the different pitches of people's voices are soprano, alto, tenor, baritone, and bass, arranged from the highest to lowest pitch.

**pixel.** Picture element. Synonym for pel.

## pointer • program

**pointer.** A variable that holds the address of a data object or function.

**pointer class.** A class that implements pointers.

**pointer to member.** An operator used to access the address of nonstatic members of a class.

**polymorphic function.** A function that can be applied to objects of more than one data type. C++ implements polymorphic functions in two ways:

1. Overloaded functions (calls are resolved at compile time)
2. Virtual functions (calls are resolved at run time)

**polymorphism.** The technique of taking an abstract view of an object or function and using any concrete objects or arguments that are derived from this abstract view.

**portability.** The ability of a programming language to compile successfully on different operating systems without requiring changes to the source code.

**positioning property.** The property of an element that is used to position the element in a collection. For example, the value of the key may be used as the positioning property.

**precedence.** The priority system for grouping different types of operators with their operands.

**precondition.** A condition that a function requires to be true when it is called.

**predefined macros.** Frequently used routines provided by an application or language for the programmer.

**predicate function.** A function that returns an IBoolean value of *true* or *false*. (IBoolean is an integer-represented Boolean type.)

**preemptive multitasking.** The operating system's ability to interrupt a thread at (almost) any time and assign the processor to a waiting thread. Multiple applications can thus run simultaneously, and a single application cannot control all of the system resources.

Contrast with nonpreemptive multitasking.

**preparation.** Any activity that the source performs before rendering the data. For example, the drag item may require that the source create a secondary thread for the source rendering to take place in. The system remains responsive to users so that they can do other tasks.

**preprocessor.** A phase of the compiler that examines the source program for preprocessor statements, which are then executed, resulting in the alteration of the source program.

**preroll.** To prepare a device to begin a playback or recording function with minimal delay.

**primary expression.** A literal, name, or name qualified by the ::(scope resolution) operator.

**primary thread.** The first thread created when a process is initialized.

See also thread.

**primitive.** See graphic primitive.

**primitive attribute.** A specifiable characteristic of a graphic primitive. See graphic attributes.

**priority queue.** A queue that has a priority assigned to its elements. When accessing elements, the element with the highest priority is removed first. A priority queue has a largest-in, first-out behavior.

**private.** Pertaining to a class member that is accessible only to member functions and friends of that class.

**process.** (1) A collection of code, data, and other system resources, including at least one thread of execution, that performs a data processing task. (2) A running application, its address space, and its resources. (3) An instance of a running program. A Win32 process owns a 4-GB address space containing the code and data for an application's .exe file; it does not execute anything. It also owns certain resources, such as files, dynamic memory allocations, and threads.

**profiling.** The process of generating a statistical analysis of a program that shows processor time and the percentage of program execution time used by each procedure in the program.

**program.** (1) One or more files containing a set of instructions conforming to a particular

## Program Files •reference class

programming language syntax. (2) A self-contained, executable module. Multiple copies of the same program can be run in different processes.

**Program Files.** A folder, which Windows 95 Setup places off the root of the drive on which Windows 95 is installed. It makes a subfolder called Accessories and places files such as WordPad and Paint there. This is the recommended default location for application programs.

**program group.** In Windows NT, a window displaying a group of programs.

**project.** (1) A container that groups related objects (tasks) into a primary window. When the user opens the object, the object has its own primary window. (2) In WorkFrame, the complete set of data objects (called project parts) and actions needed to build a single target, such as a dynamic link library (DLL) or executable file (EXE).

**Project Access Method (PAM).** A dynamic link library that contains a set of methods through which a simple abstraction of a file system or repository is provided to WorkFrame. PAMs enable a WorkFrame project to contain any kind of object that a PAM can support, for example a version of a file in a source control library, or another file system like MVS or AIX.

**project hierarchy.** A project tree that represents dependencies between projects. The WorkFrame project paradigm requires that one project should be created for every target. Dependencies between projects and their targets should be expressed in a project hierarchy. That is, if a project's build depends on the target of another project, the dependent project should contain the project it depends on. The dependent project is then said to *nest* the other project. This enables the Build tool to perform Builds in a depth-first search manner from anywhere in the project hierarchy.

**project-scoped action.** An action that applies to a project as a whole, or to a project's specially designated parts. Specially designated project parts are the project's make file and target. An example of a project-scoped action is Debug, which is invoked on the project's target.

**Project Smarts.** A project catalog that contains templates for common types of applications.

**Project Smarts application.** A skeletal application that consists of template source code and a configured project revolving around an application theme. It serves as a starting point for similar applications.

**property function.** A function that is used to determine whether the element it is applied to has a given property or characteristic. A property function can be used, for example, to remove all elements with a given property.

**protected.** Pertaining to a class member that is only accessible to member functions and friends of that class, or to member functions and friends of classes derived from that class.

**prototype.** A function declaration or definition that includes both the return type of the function and the types of its arguments.

**public.** Pertaining to a class member that is accessible to all functions.

**pure virtual function.** A virtual function that has a function initializer of the form `= 0;`.

## Q

**qualified name.** Used to qualify a nonclass type name such as a member by its class name.

**queue.** A sequence with restricted access in which elements can only be added at the back end (or bottom) and removed from the front end (or top). A queue is characterized by first-in, first-out behavior and chronological order.

## R

**redirection.** In the shell, a method of associating files with the input or output of commands. *X/Open*.

**reentrant.** The attribute of a program or routine that allows the same copy of a program or routine to be used concurrently by two or more tasks.

**reference class.** A class that links a concrete class to an abstract class. Reference classes make

## register storage class specifier •secondary window

polymorphism possible with the Collection Classes.

**register storage class specifier.** A specifier that indicates to the compiler within a block scope data definition, or a parameter declaration, that the object being described will be heavily used.

**regular expression.** (1) A mechanism to select specific strings from a set of character strings. (2) A set of characters, meta-characters, and operators that define a string or group of strings in a search pattern. (3) A string containing wildcard characters and operations that define a set of one or more possible strings.

**regular file.** A file that is a randomly accessible sequence of bytes, with no further structure imposed by the system. *X/Open. ISO.1.*

**relation.** An unordered flat collection class that uses keys, allows for duplicate elements, and has element equality.

**renderer.** An object that renders data using a particular mechanism, such as using files or shared memory. It contains definitions of supported rendering mechanisms and formats and types. Renderers are maintained positionally (1-based).

**rendering.** The transfer or re-creation of the dragged object from the source window to the target window.

**rendering format.** Identifies the actual format of the data being rendered in a direct manipulation operation.

**rendering mechanism.** Identifies the actual format of the data being rendered in a direct manipulation operation.

**resource file.** A file that contains data used by an application, such as text strings and icons.

**response file.** A file that acts as an extension of command-line input, providing options and parameters as input to a component.

**return.** A language construct that ends an execution sequence in a procedure. *IBM.*

**returned element.** An element returned by a function as the return value.

**RMFs.** Rendering mechanisms and formats.

**root.** A node that has no parent. All other nodes of a tree are descendants of the root.

**RTTI.** Run-time type identification.

**run-time library.** A compiled collection of functions whose members can be referred to by an application program during run-time execution. Typically used to refer to a dynamic library that is provided in object code, such that references to the library are resolved during the linking step. The run-time library itself is not statically bound into the application modules.

**run-time type identification (RTTI).** A mechanism in the C++ language for determining the class of an object at run time. It consists of two operators, one for determining the run-time type of an object (**typeid**) and one for doing type conversions that are checked at run time (**dynamic\_cast**). A **type\_info** class describes the RTTI available and defines the type returned by the **typeid** operator.

## S

**SBCS (Single-Byte Character Set).** See single-byte character set.

**scalar.** An arithmetic object, or a pointer to an object of any type.

**scan.** To search backward and forward at high speed on a CD audio device. Scanning is analogous to fast forwarding.

**scope.** That part of a source program in which an object is defined and recognized.

**scope operator (::).** An operator that defines the scope for the argument on the right. If the left argument is blank, the scope is global; if the left argument is a class name, the scope is within that class. Also called a scope resolution operator.

**scroll increment.** The number by which the current value of the circular slider is incremented or decremented when a user presses one of the circular slider control buttons.

**secondary window.** See *child window*.

## semaphore •source type

**semaphore.** A synchronization kernel object used for resource-counting. A semaphore offers a thread the ability to query the number of resources available. If one or more resources are available, the count of available resources is decremented.

See also critical section, mutex, and event.

**sequence.** A sequentially ordered flat collection.

**sequential collection.** An abstract class with the property of sequentially ordered elements.

**server.** In Compound Document Framework, an application or a document component that supplies an object. For example, a drawing program that provides a picture that can be placed inside a word processing document is referred to as a server.

**shell.** A program that interprets sequences of text input as commands. It may operate on an input stream or it may interactively prompt and read commands from a terminal. *X/Open*.

This feature is provided as part of OpenEdition MVS Shell and Utilities feature licensed program.

**siblings.** All the children of a node are said to be siblings of one another.

**signal.** (1) A condition that may or may not be reported during program execution. For example, SIGFPE is the signal used to represent erroneous arithmetic operations such as a division by zero. (2) A mechanism by which a process may be notified of, or affected by, an event occurring in the system. Examples of such events include hardware exceptions and specific actions by processes. The term *signal* is also used to refer to the event itself. *X/Open*. *ISO.1*. (3) In AIX operating system operations, a method of interprocess communication that simulates software interrupts. *IBM*.

**signaled.** A state in which an object has been reactivated after the threads have been put to sleep. For example, if a thread in a parent process needs to wait for the child process to terminate, the parent's thread puts itself to sleep until the kernel object identifying the child process becomes signaled.

**signal handler.** A function to be called when the signal is reported.

**silent mode.** See unattended mode.

**single-byte character set (SBCS).** A set of characters in which each character is represented by a 1-byte code.

**slash.** The character /, also known as *solidus*. This character is named <slash> in the portable character set.

**SMPTE time code.** A frame-numbering system developed by SMPTE that assigns a number to each frame of video. The 8-digit code is in the form HH:MM:SS:FF (hours, minutes, seconds, frame number). The numbers track elapsed hours, minutes, seconds, and frames from any chosen point.

**sorted bag.** A sorted flat collection that allows duplicate elements.

**sorted collection.** (1) An abstract class with the property of sorted elements. (2) In general, any collection with sorted elements.

**sorted map.** A sorted flat collection with key and element equality.

**sorted relation.** A sorted flat collection that uses keys, has element equality, and allows duplicate elements.

**sorted set.** A sorted flat collection with element equality.

**source directory.** A directory where a project's parts are physically stored. A project may have many source directories.

**source file.** A file that contains source statements for such items as high-level language programs and data description specifications. *IBM*.

**source program.** A set of instructions written in a programming language that must be translated to machine language before the program can be run. *IBM*.

**source type.** A source type appears in an action's list of source types. An action's list of source types specifies the kind of parts or files to which the action applies.

## space character •structure tag

**space character.** The character defined in the portable character set as <space>. The space character is a member of the space character class of the current locale, but represents the single character, and not all of the possible members of the class. *X/Open*.

**specifiers.** Used in declarations to indicate storage class, fundamental data type and other properties of the object or function being declared.

**sprite.** A small graphic that can be moved independently around the screen, producing animated effects.

**stack.** A data structure in which new elements are added to and removed from the top of the structure. A stack is characterized by Last-In-First-Out (LIFO) behavior.

**stack frame.** The physical representation of the activation of a routine. The stack frame is allocated and freed on a LIFO (last in, first out) basis.

**stack storage.** Synonym for *automatic storage*.

**standard error.** An output stream usually intended to be used for diagnostic messages.

**standard input.** An input stream usually intended to be used for primary data input. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command.

**standard output.** An output stream usually intended to be used for primary data output. *X/Open*. When programs are run interactively, standard output usually goes to the display unless redirection or piping is used, in which case standard output can go to a file or to another command.

**statement.** An instruction that ends with the character ; (semicolon) or several instructions that are surrounded by the characters { and }.

**static.** A keyword used for defining the scope and linkage of variables and functions. For internal variables, the variable has block scope and retains its value between function calls. For external values, the variable has file scope and

retains its value within the source file. For class variables, the variable is shared by all objects of the class and retains its value within the entire program.

**step backward.** In multimedia applications, to move the medium backward one frame or segment at a time.

**step forward.** In multimedia applications, to move the medium forward one frame or segment at a time.

**step frame.** A function of devices such as digital video and videodisc players that enables a user to move frame-by-frame in either direction.

**storage class specifier.** One of: auto, register, static, or extern.

**stream.** (1) A continuous stream of data elements being transmitted, or intended for transmission, in character or binary-digit form, using a defined format. (2) A file access object that allows access to an ordered sequence of characters, as described by the ISO C standard. A stream provides the additional services of user-selectable buffering and formatted input and output.

**stream buffer.** A stream buffer is a buffer between the ultimate consumer, ultimate producer, and the I/O Stream Library functions that format data. It is implemented in the I/O Stream Library by the streambuf class and the classes derived from streambuf.

**string.** A contiguous sequence of characters.

**string literal.** Zero or more characters enclosed in double quotation marks.

**structure.** A construct that contains an ordered group of data objects. Unlike an array, the data objects within a structure can have varied data types.

**structured exception handling.** A Windows-specific mechanism for handling system exceptions that matches exceptions with handlers based on the value returned from an exception filter expression.

**structure tag.** The identifier that names a structure data type.

## subclass •thread-local storage

**subclass.** See derived class.

**subscript.** One or more expressions, each enclosed in brackets, that follow an array name. A subscript refers to an element in an array.

**subsystem.** A secondary or subordinate system, usually capable of operating independently of or asynchronously with, a controlling system. *ISO Draft.*

**subtree.** A tree structure created by arbitrarily denoting a node to be the root node in a tree. A subtree is always part of a whole tree.

**superclass.** See base class and abstract class.

**superset.** Given two sets A and B, A is a superset of B if and only if all elements of B are also elements of A. That is, A is a superset of B if B is a subset of A.

**switch statement.** A C or C++ language statement that causes control to be transferred to one of several statements depending on the value of an expression.

**system default.** A default value defined in the system profile. *IBM.*

## T

**table type.** A category of database table, such as user table, system catalog table, or view.

**tabular implementation.** An implementation that stores the location of elements in tables. Elements in a tabular implementation are accessed by using indices to arrays.

**tabular sequence.** A sequence that uses a tabular implementation.

**target.** A WorkFrame project's target is the file that is produced as a result of a project build.

**task.** (1) In a multiprocessing or multiprogramming environment, one or more sequences of instructions treated by a control program as an element of work to be accomplished by a computer. *ISO-JTC1. ANSI.* (2) A routine that is used to simulate the operation of programs. Tasks are said to be *nonpreemptive* because only a single task is

executing at any one time. Tasks are said to be *lightweight* because less time and space are required to create a task than a true operating system process.

**taskbar.** In Windows 95, a bar at the bottom (default position) of the desktop that shows all open applications and active windows. Clicking on any task button brings the corresponding session to the foreground.

**task library.** A class library that provides the facilities to write programs that are made up of tasks.

**template.** A family of classes or functions where the code remains invariant but operates with variable types.

**terminals.** Synonym for *leaves*.

**template class.** A class instance generated by a class template.

**template function.** A function generated by a function template.

**text file.** A file that contains characters organized into one or more lines. The lines must not contain null characters and none can exceed {LINE\_MAX}—which is defined in limits.h—bytes in length, including the new-line character. The term *text file* does not prevent the inclusion of control or other nonprintable characters (other than NUL). *X/Open.*

**this.** A C++ keyword that identifies a special type of pointer in a member function, one that references the class object with which the member function was invoked.

**this collection.** The collection to which a function is applied.

**thread.** (1) The smallest unit or path of execution within a process. *IBM.* (2) A piece of executing code. (3) In Windows, each thread is allocated its own stack from the owning process' 4-GB address space, and each one has its own set of processor registers, called the thread's context. See also primary thread and zero page thread.

**thread-local storage.** A Windows-specific mechanism that allows each thread in a

## thread synchronization •typed implementation class

multithreaded process to allocate storage for its corresponding data.

**thread synchronization.** The ability to synchronize the activities of various threads. A thread synchronizes itself with another thread by putting itself to sleep. Before doing so, the thread notifies the operating system as to what event has to occur in order for the thread to resume execution.

**throw expression.** An argument to the C++ exception being thrown.

**tic.** Approximately 838 nanoseconds. A tic is the smallest unit of time measured by the Performance Analyzer counter.

**tilde.** The character ~. This character is named <tilde> in the portable character set.

**time code.** See SMPTE time code.

**time on the stack.** Total time a call is on the call stack while its thread is executing.

**tool bar.** The area under the title bar that displays the tools available.

**token.** The smallest independent unit of meaning of a program as defined either by a parser or a lexical analyzer. A token can contain data, a language keyword, an identifier, or other parts of language syntax.

**transparency.** Refers to when a selected color on a graphics screen is made transparent to allow the video behind it to become visible.

**transparent color.** (1) A clear color used to indicate the part of the bitmap that is not drawn for the bitmap. The area under the bitmap is not overpainted for areas of the bitmap that are set to the transparent color. (2) Video information is considered as being present on the video plane that is maintained behind the graphics plane. When an area on the graphics plane is painted with a transparent color, the video information in the video plane is made visible.

**trap.** An unprogrammed conditional jump to a specified address that is automatically activated by hardware. A recording is made of the location from which the jump occurred. *ISO-JTC1*.

**treble.** (1) The upper half of the whole vocal or instrumental tonal range. (2) The higher portion of the audio frequency range in sound recording.

**tree.** A hierarchical collection of nodes that can have an arbitrary number of references to other nodes. A unique path connects every two nodes.

**tree control.** A type of control that shows the hierarchical relationships among a set of objects by indenting them as in an outline. The user can expand or collapse the various branches (levels) of the tree (outline).

**true and additional.** The most accurate or most descriptive (primary) type of an object (true) and the other or secondary types (additional). For example, if the object is a text file, its true type is text; if the file was a C source code file, its true type is C code.

**try block.** (1) A block in which a known C++ exception is passed to a handler. (2)

**type.** (1) The description of the data and the operations that can be performed on or by the data. See also *data type*. (2) In WorkFrame, describes a group of project files of parts in terms of an expression, such as file masks, regular expressions, or a list of other types, logical-OR'd.

**type balancing.** A conversion that makes both operands the same data type. If the operands do not have the same size data type, the compiler converts the value of the operand with the smaller type to a value having the larger type.

**type class.** In WorkFrame, represents the method by which an object is determined to be a member of a type. "File mask" is an example of a type class. Membership to a "file mask" type is determined by matching the file mask filter to the object's name. Other examples of type classes are "regular expression," and "PAM name," where the named Project Access Method determines membership to a type.

**type definition.** A definition of a name for a data type. *IBM*.

**typed implementation class.** A class that implements a concrete class and provides an interface that is specific to a given element type. This interface allows the compiler to verify that,



## typeless implementation class •videocassette recorder (VCR)

for example, integers cannot be added to a set of strings.

**typeless implementation class.** A class that implements a concrete class and provides an interface that is not specific to a given element type.

**type specifier.** Used to indicate the data type of an object or function being declared.

## U

**undefined behavior.** Referring to a program or function that may produce erroneous results without warning because of its use of an indeterminate value, or because of erroneous program constructs or erroneous data.

**ultimate consumer.** The target of data in an I/O operation. An ultimate consumer can be a file, a device, or an array of bytes in memory.

**ultimate producer.** The source of data in an I/O operation. An ultimate producer can be a file, a device, or an array of bytes in memory.

**unary expression.** An expression that contains one operand.

**unattended mode.** A mode in which no operator is present or in which no operator station is included at system generation.

**unbounded collection.** A collection that has no upper limit on the number of elements it can contain.

**undefined behavior.** Referring to a program or function that may produce erroneous results without warning because of its use of an indeterminate value, or because of erroneous program constructs or erroneous data.

**undefined cursor.** A cursor that may or may not be valid, and that may or may not refer to a different element of the collection from the element it referred to before the function call that resulted in it becoming undefined. An undefined cursor may refer to no element of the collection, and still be a valid cursor.

**underflow.** (1) A condition that occurs when the result of an operation is less than the smallest

possible nonzero number. (2) Synonym for arithmetic underflow, monadic operation.*IBM.*

**union.** (1) Structures that can contain different types of objects at different times. Only one of the member objects can be stored in a union at any time. *IBM.* (2) Given the sets A and B, all elements of A, B, or both A and B.

**union tag.** The identifier that names a union data type.

**unique collection.** A collection in which the value of an element only occurs once; that is, there are no duplicate elements.

**unload.** To eject the medium from the device.

**unordered collection.** A collection that has no order to its elements.

**unrecoverable error.** An error for which recovery is impossible without use of recovery techniques external to the computer program or run.

**user object.** An object, such as an icon, a window, a menu, or an accelerator table. In Windows, most user objects are owned by the thread that created them (icons, cursors, and windows classes are owned by a process).

## V

**variable.** In programming languages, a language object that may take different values, one at a time. The values of a variable are usually restricted to a certain data type. *ISO-JTC1.*

**VGA.** Video graphics adapter.

**video.** Pertaining to the portion of recorded information that can be seen.

**video attributes.** The standard video attributes are: brightness, contrast, freeze, hue, saturation, and sharpness.

**video graphics adapter (VGA).** A graphics controller for color displays. The pel resolution of the video graphics adapter is 4:4.

**videocassette recorder (VCR).** A device for recording or playing back videocassettes.

## videodisc • write

**videodisc.** A disc on which programs have been recorded for playback on a computer or a television set; a recording on a videodisc. The most common format in the United States and Japan is an NTSC signal recorded on the optical reflective format.

**videodisc player.** A device that provides video playback for prerecorded videodiscs.

**view.** (1) Displayed contents of a document. In the Editor, multiple views of the same document can be open at the same time. (2) In Compound Document Framework, the user interface controls and associated handlers for a document component. The view and the model comprise the two pieces of a document component. The Compound Document Framework provides an `IView` base class from which other view classes can be derived.

**virtual function.** A function of a class that is declared with the keyword **virtual**. The implementation that is executed when you make a call to a virtual function depends on the type of the object for which it is called. This is determined at run time.

**visible.** Visibility of identifiers is based on scoping rules and is independent of *access*.

**volatile.** An attribute of a data object that indicates the object is changeable beyond the control or detection of the compiler. Any expression referring to a volatile object is evaluated immediately, for example, assignments.

## W

**white space.** Space characters, tab characters, form feed characters, and new-line characters.

**wide character.** A character whose range of values can represent distinct codes for all members of the largest extended character set specified among the supporting locales.

**Win32.** The name of a 32-bit application programming interface (API) developed by Microsoft.

See also Win32 API.

**Win32 API.** (1) A set of Win32 functions that can be called from source code. (2) A 32-bit

version of the 16-bit Windows 3.1 API (native to Windows NT).

See also Win32.

**Win32s.** A subset of the Win32 API that can run under Windows 3.x. (The s stands for subset.) It consists of a virtual device driver and dynamic link libraries (DLLs) that add the Win32 API to the 16-bit Windows 3.x system. It includes structured exception handling and limited implementations of memory-mapped files.

See also Win32 API and Windows 95.

**Windows NT.** An operating system that the Win32 API is implemented on. It is a portable, high-end operating system, which supports multitasking. It is the only operating system that allows implementation of the Win32 APIs on machine architectures based on processors other than the x86, and it supports multiple processors.

See also Win32 API.

**Windows 95.** (1) A 32-bit operating system that allows you to run 32-bit application. Windows 95 is a multitasking, multithreaded operating system that can control multiple programs at once. Each program can have multiple concurrent threads or independently executing subcomponents. (2) A platform that the Win32 API is implemented on. It supports image color matching, modems, and other services. It partially supports asynchronous file I/O, debugging, registry, security, and event-logging functions.

**word boundary.** Any storage position at which data must be aligned for certain processing operations. The halfword boundary must be divisible by 2; the fullword boundary by 4; and the doubleword boundary by 8. *IBM*.

**working directory.** (1) Synonym for *current working directory*. (2) The directory where files that are copied or dragged into the project are stored. Actions are also executed in this directory, so this directory is where many output files are placed.

**workspace.** A container object that provides for the association and management of task-related windows within a parent window.

**write.** (1) To output characters to a file, such as standard output or standard error. Unless

otherwise stated, standard output is the default output destination for all uses of the term *write*. *X/Open*. (2) To make a permanent or transient recording of data in a storage device or on a data medium. *ISO-JTC1*. *ANSI*.

## Z

**zero page thread.** A thread with a priority level of 0 that zeros out any free pages in the system when there are no other threads that need to perform work in the system. No other threads can have a priority level of 0.

See also thread and primary thread.

zero page thread •(::) (double colon)

## Numerics

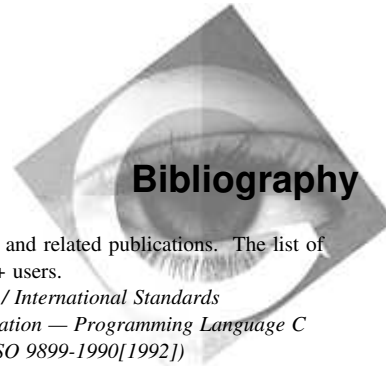
**24-bit color.** A digital standard that uses 24 bits of information to describe each color pixel, providing up to 16.7 million colors in one image (the highest digital standard currently available).

**8-bit color.** A digital standard that uses 8 bits of information to describe each color pixel, providing up to 256 colors in one image (the standard for VGA displays).

## Special Characters

**(::) (double colon).** Scope operator. An operator that defines the scope for the argument on the right. If the left argument is blank, the scope is global; if the left argument is a class name, the scope is within that class. Also called a scope resolution operator.





This bibliography lists the publications that make up the IBM VisualAge for C++ library and related publications. The list of related publications is not exhaustive but should be adequate for most VisualAge for C++ users.

---

## The IBM VisualAge for C++ Library

The following books are part of the IBM VisualAge for C++ library.

- *Installation Guide and Product Overview*, S33H-5030
- *User's Guide*, S33H-5031
- *Programming Guide*, S33H-5032
- *Visual Builder User's Guide*, S33H-5034
- *Visual Builder Parts Reference*, S33H-5035
- *Building VisualAge for C++ Parts for Fun and Profit*, S33H-5036
- *Open Class Library User's Guide*, S33H-5033
- *Open Class Library Reference*, S33H-5039
- *Language Reference*, S33H-5037
- *C Library Reference*, S33H-5038
- *SOM Programming Guide*, S33H-5044
- *SOM Programming Reference*, S33H-5044

---

## C and C++ Related Publications

- *Portability Guide for IBM C*, SC09-1405
- *American National Standard for Information*

*Systems / International Standards  
Organization — Programming Language C  
(ANSI/ISO 9899-1990[1992])*

## Non-IBM Publications

Many books have been written about the C++ language and related programming topics. The authors use varying approaches and emphasis. The following is a sample of some non-IBM C++ publications that are generally available. This sample is not an exhaustive list. IBM does not specifically recommend any of these books, and other C++ books may be available in your locality.

- *The Annotated C++ Reference Manual* by Margaret A. Ellis and Bjarne Stroustrup, Addison-Wesley Publishing Company.
- *C++ Primer* by Stanley B. Lippman, Addison-Wesley Publishing Company.
- *Object-Oriented Design with Applications* by Grady Booch, Benjamin/Cummings.
- *Object-Oriented Programming Using SOM and DSOM* by Christina Lau, Van Nostrand Reinhold.



## Special Characters

- command for NMAKE 734
- ! command for NMAKE 735
- /? compiler option 171
- /? linker option 202
- /? option for ILIB 529
- ? resource compiler option 616
- /Fb 389
- @ command for NMAKE 735
- \ (continuation character) 69

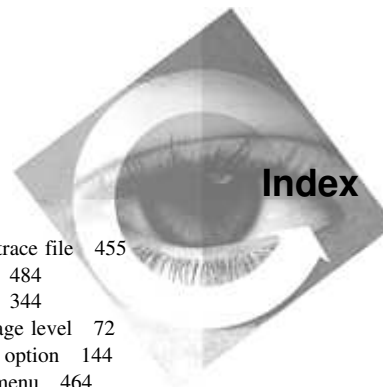
## A

- A option for NMAKE 713
- abstract code units (ACUs)
  - example 86
- Accelerator editor (resources) 665
  - customizing 668
- accelerator tables 657
  - adding accelerators 666
  - editing scripts 665
- accelerators 665—668
  - changing keys 667
  - changing the identifier of 666
  - checking for duplicate keys 668
  - copying, moving or removing 667
  - source code 668
  - viewing 665
- access violations 512
- accessing logical keys not on keyboard 252
- action status bar (Browser) 341, 352
- ACUs (abstract code units)
  - example 86
- adding menu items (Browser) 393
- adding or replacing objects in a library,
  - ILIB 526
- address breakpoint 283
- aiding program understanding 378
- align data items in structures and unions 147
- /ALIGNMENT linker option 203
- Analyze Trace window 472
  - check boxes 473
  - entry field 473
  - push buttons 473

- analyzing a trace file 455
- annotations 484
- anonymous 344
- ANSI language level 72
  - compiler option 144
- application menu 464
- Application Monitor window 470
  - push buttons 470
  - status area 470
- arcs, Dynamic Call Graph 487, 488
- assembler code, calling 159
- assembler listing file 75
  - compiler option 122
- assigning frequently used commands to keys 252
- assigning menu selections to keys 252
- assisting in development 375
- attach to processes 268
- attributes, text (Browser) 347, 349
- auto-inlining 86
  - limiting memory used during 164

## B

- /B compiler option 105, 172
- B option for CPPFILT 759
- backslash (\) 69
- /BACKUP option for ILIB 529
- /BASE linker option 203
- BASE module statement 536
- bitmap dimensions (Browser) 360
- block of text
  - rectangular 237
- .BMP files 627
- breakpoint list 286
- Breakpoint List window 316
- breakpoints menu 279
- BROWSE link 336
- /BROWSE linker option 204
- /BROWSE option for ILIB 530
- browser
  - adding menu items 393
  - aiding program understanding 378
  - assisting in development 375
  - compiler option 123
  - database 330, 333, 337, 392



- browser (*continued*)
  - database search 372
  - ending 336
  - fast-path keys 415
  - generated flags 344
  - Graph window 351
  - help levels 365
  - history 373
  - linker option 204
  - List window 340
  - objects 330, 343
  - options for starting 333
  - overview 329
  - paths 363
  - popup menus 424
  - pulldown menus 416
  - settings 363
  - starting from the command line 333
  - starting from the Debugger 334
  - starting from the Program Manager 334
  - starting from the WorkFrame 334
  - tour 395
  - troubleshooting 413
  - updating database 392
  - user interface 339
- Browser objects 356, 425
- browsing 330
  - compiler generated Browser (.PDB)
    - files 329
    - more than one file 377
  - Open Class Library classes 376
  - program files (.DLL,.EXE,.LIB) 329
  - without recompiling 376
  - WorkFrame projects 329
- BRS files 368
- brsmenu.txt (Browser) 393
- BSS32 segment
  - renaming 161
- buffer wrap 450
- Build Smarts 34
- build utility
  - See* building project targets
- building project targets 37
  - build actions 40
  - build options notebook 39
    - actions page 40
    - display page 46
    - make page 42
    - project page 44
  - building from the command line 48

- building project targets (*continued*)
  - command line syntax 48
  - comparing make and build 37
  - concurrency 37
  - dependency file 42
  - displaying MakeMake window 46
  - locking projects 37
  - make processing options 42
  - make utility 43
  - makefile generation script 43
  - NMAKE make utility 43
  - prerequisites for building 38
  - prompting during builds 46
- built-in functions 86

## C

- /C compiler option 172
- C language
  - See also Language Reference*
  - allowing double slashes as comments 148
  - restricted compiler options 113
- C option for CPPFILT 754
- C option for LOCALDEF 610
- C option for NMAKE 713
- C++ language
  - See also Language Reference*
  - compatibility with older versions 73
  - resolving templates 126
  - restricted compiler options 114
- call depth, selecting 449
- Call Nesting diagram 479
  - menu bar 480
  - pop-up menus 483
  - status area 483
- Call Stack Window 305
- calling conventions
  - #pragma linkage** 90
  - compiler options 160
  - default 90
  - linkage keywords 90
  - setting 90
- calls 434
  - as basis for arc color 488
  - displaying number of 493
  - in Call Nesting 479
  - maximum number made 488
  - order 478
  - shown as arcs 486
  - to PERF entry point 435



- calls (*continued*)
  - to PerfStart/PerfStop entry points 436
  - with pattern recognition 477
  - with triggers 465
- Cancel push button 463, 473
- cascaded menus 659, 664
- case sensitivity
  - compiler options 107
- \_\_cdecl** calling convention
  - compiler option 160
  - keyword 90
  - setting 90
- change address breakpoint 283
- changing
  - fonts (Browser) 366
- char, treatment when unspecified 143
- character map file, specifying for
  - LOCALDEF 610
- characters
  - backslash (\) 109
  - continuation (\) 69
- check boxes 636
  - properties, setting 647
- check heap when stopping 287
- choosing runtime libraries 91
- class execution time 488
- Class Information window 494
- class objects (Browser) 343
- close debugger 279
- closing the Browser 336
- code generation options, compiler 154
- code set conversion utilities
  - ICONV 605
  - ICONVDEF 607
- CODE32 segment
  - renaming 161
  - setting attributes of 161
- color
  - Browser Graph window 358
  - Browser List window 347
- color, node and arc 488
- combining C and C++ files 63
- combo boxes 636
  - properties, setting 650
- command file, using to start Performance Analyzer 439
- command files for NMAKE 711
- command line
  - help for linker 202, 209
  - help from 60, 171
- command line (*continued*)
  - linking from 180
  - precedence over ICC options 108, 112
  - setting compiler options on 107
  - setting linker options on 197
- command line parameters for ILIB 520
- command line processor, conversion utility 765
- comments in C files 148
- Compatible language level 73
  - compiler option 144
- compiler 57
  - environment variables 66
  - input 63
  - limit working set size 164
  - listing files
    - See* listing files, compiler
  - listings 79
  - logo display 173
  - messages 80
  - output 75
  - output file options 121
  - passing options to linker 172
  - response files 61
  - return codes 80
  - starting the 59
  - temporary files 79
  - work files 79
- compiler errors
  - controlling level and number 80
  - intermediate code linker 84
  - set maximum number of 135
- compiler gen 345
- compiler options 117
  - accumulation of 111
  - classification by function 116
  - combinations for specifying libraries 115
  - conflicting 112
  - for code generation 154
  - for debugging and diagnostics 135
  - for include file 128
  - for listing files 130
  - for optimizing 162
  - for output file management 121
  - for PM programming 114
  - for preprocessing 151
  - for source code 143
  - language-dependent 113
  - list of 117
  - online listing 60
  - parameters 109

- compiler options (*continued*)
  - precedence 108, 112
  - related 112
  - scope 111
  - setting 107
    - in ICC 107
    - in WorkFrame 108
    - on the command line 107
  - specifying 107
  - switches (+|-) 110
  - syntax for xxxiv
  - with multiple source files 111
- compiler source files
  - compiling multiple 63
  - default file extensions 64
  - double slashes in C files 148
  - file types 64
  - ignoring columns in 145, 148
  - ignoring sequence numbers in 148
  - include source code in listing file 134
  - information about in listing 130
  - language level 71
  - line numbers in map file 210
  - mixed C and C++ 63
  - naming in ICC 67
  - options with multiple 111
  - organizing for precompiled headers 100
  - sequence numbers in 148
  - setting default extension 144
  - setting margins of 145
  - source code compiler options 143
  - specifying C files 149
  - specifying C++ files 150
- compiler-generated symbols, demangling 756, 761
- compiling 57
  - a .dll file 155
  - an .exe file 155
  - for Performance Analyzer 156
  - for the debugger 263
  - from a makefile 62
  - from the command line 60
  - improving compile time 94
  - invoking the linker 182
  - mixed C and C++ files 63
  - multiple source files 63
  - set maximum number of errors 135
  - syntax check only 123
  - using response files 61
  - within WorkFrame 59
- compiling (*continued*)
  - without linking 59, 172
- compiling (Browser) 335
- compiling and linking
  - See also* compiler options
  - C and C++ files 63
  - environment variables 66
  - for the debugger 263
  - multiple source files 111
- compiling and linking programs 433
- components box 276
- components, enabling/disabling 447
  - displaying information 447
  - shown as nodes 447
  - viewing/hiding 447
- conflicting compiler options 112
- Console window 59
- CONST32\_RO segment
  - default attributes 161, 213
  - renaming 161
  - setting attributes of 161
- CONSTANT attribute 538
- constant segment
  - default attributes 161, 213
  - renaming 161
  - setting attributes of 161
- container view 330
- container view (Browser) 341
  - ordering 342
- context switches 479
- continuation character (\) 69
- Control Window 275
- controlling compiler input 63
- controlling compiler messages 138
- controlling message severity 138
- controlling size of enum variables 149
- converting character set definitions 605
- converting code sets 605
- correlation, understanding 477
  - in Call Nesting 477
  - in Execution Density 477
  - in Time Line 477
- CPPFILT utility
  - binary mode
    - example of output 758
    - options 758—762
    - output 757
    - syntax 757
  - demangling compiler-generated symbols 756, 761

- CPPFILT utility (*continued*)
  - demangling public symbols 760
  - demangling stand-alone class names 754
  - displaying help 755
  - generating NONAME keyword for exports 759
  - generating ordinals 760
  - including exported symbols 762
  - including referenced (EXTDEF) symbols 761
  - producing both demangled and mangled names 756
  - producing symbol map 755
  - removing leading underscore 762
  - specifying binary mode 759
  - specifying field width 756
  - suppress comments in binary mode 762
  - suppressing logo 755, 761
  - text mode
    - options 754—756
    - output 754
    - syntax 754
    - what it does 753
- Create Trace window 462
  - entry fields 463
  - push buttons 463
- creating conversion tables 607
- creating locale files 609
- cross-reference table in listing file 135
- cumulative compiler options 70, 111
- .CUR files 627
- current column indicator 504
- customizing
  - autosave 253
  - editor 249
  - fonts 256
  - keyboard 252
  - menu bar 251
  - tab settings 255
  - token attributes 257
  - tool bar 250
- customizing Resource Workshop 628

## D

- /D compiler option 151
- D option for NMAKE 713
- d resource compiler option 615
- Data Access Builder
  - attributes page, Settings notebook 574

- Data Access Builder (*continued*)
  - class mapping, customizing 573
  - code generation options, selecting 561
  - code generation options, specifying defaults 563
  - columns page, Settings notebook 569
  - connecting to database tables 559
  - creating classes, overview of steps 557
  - data identifiers and primary keys 576
  - database definition changes 571
  - database tables/views, accessing 559
  - deleting class and table objects 567
  - foreign keys page, Settings notebook 570
  - generated files, using 580
  - generated files, viewing 580
  - generating multiple classes 566
  - generating source code 578
  - icons, customizing size 556
  - icons, identifying 565
  - mappings, customizing the display 556
  - members page, Settings notebook 577
  - members section, navigating 578
  - migrating a previously saved session 554
  - names page, Settings notebook 573
  - opening a previously saved session 553
  - overview 551
  - primary keys 576
  - Project Smarts, setting up a project 552
  - registering database products 556
  - save as 579
  - saving a session 579
  - source code, generating 578
  - source code, using 580
  - source code, viewing 580
  - starting 553
  - Startup window 555
  - supported database products 552
  - table page, Settings notebook 568
  - table/view Settings notebook 568
  - using the generated files 580
  - view page, Settings notebook 568
  - viewing generated files 580
  - window, identifying the parts 554
  - WorkFrame, setting up a project 552
- data items, aligning structures and unions in 147
- data types, valid 325
- data, filtering 475
  - displaying 493
- DATA32 segment
  - renaming 161

- DATA32 segment (*continued*)
  - setting attributes of 161
- database, Browser 330, 333, 335, 337, 392
- DB2
  - See* ODBC
- DBCS (double-byte character set) 147
- /DBGPACK linker option 205
- ddnames, allowing use of 146
- /DEBUG linker option 206
- debug on demand 271
- debugger 263, 334
  - compiler options 136, 137
  - generating information for 77
  - information, packing 205
  - linker option 77, 205, 206
  - optimization with 77
  - starting the Browser from 334
- debugger search path 266, 268
- debugging 265
  - compiler option for 136
  - information, packing 205
  - line-number only 137
  - linker option 205, 206
  - memory management 137
- debugging applications on Windows NT or Windows 95 265
- debugging compiler options 135
- debugging options, compiler 135
- debugging windows 275
- DECORATED attribute 538
- /DEF option for ILIB 530
- default
  - attributes of segments 213
  - calling convention 90
  - changing, for compiler input files 144
  - compiler executable output 155
  - extension for executable 212
  - file extensions 64
  - language of compiler files 150
  - libraries, searching 206
  - library information, suppressing 158
  - linker file names 189
  - name for executable 212
  - precompiled header names 98
  - runtime library 91
  - stack size 105
- default code and text segments
  - renaming 161
  - setting attributes of 161
- default data segment
  - renaming 161
  - setting attributes of 161
- /DEFAULTLIB linker option 93, 94, 206
- defer breakpoint 281
- defining preprocessor macros 151
- delete all breakpoints 286
- demangling C++ names
  - compiler-generated symbols 756, 761
  - exported symbols 762
  - object (.lib) and library (.lib) files 757
  - public symbols (PUBDEF, COMDAT, COMDEF) 760
  - referenced (EXTDEF) symbols 761
  - stand-alone classes 754
  - text files 753
- description files for NMAKE
  - creating 727
  - description blocks 715
  - format 716
  - macros 718
- DESCRIPTION module statement 537
- description, Performance Analyzer 431
- detach processes 268
- Details pane 509
- DGROUP 161, 539
- diagnosing errors, Call Nesting 479
- diagnostic compiler options 135
- diagrams, using to analyze 455
  - Call Nesting 479
  - displaying 456
  - Dynamic Call Graph 485
  - Execution Density 499
  - Statistics 505
  - Time Line 511
  - time-scaled 477
- dialog boxes 631
  - adding menus 633
  - controls 636
    - adding 636
    - aligning 641
    - custom 637
    - grouping 640
    - moving 638
    - resizing 638
    - selecting 637
    - tabbing through 640
- editing scripts 631
- frames 634, 636
- properties, setting 632—635, 644—652

- dialog editor (resources) 631
  - Alignment palette 632, 642
  - customizing 654
  - Duplicate tool 643
  - Properties window 644
  - Selector tool 637
  - Set Groups tool 640
  - Set Order tool 641
  - starting 631
  - Style window 645—652
  - Test tool 653
  - Tools palette 632, 636
    - adding custom controls 637
  - undoing actions 644
- dialogs
  - Browser Files 390
  - Find (Browser) 371
  - Font (Browser) 366
  - History (Browser) 373
  - Load Database (Browser) 368
  - Merge Database (Browser) 369
  - new user help (Browser) 366
  - Overview (Browser) 354
  - Print (Browser) 350, 361
  - QuickBrowse (Browser) 390
  - Save As... (Browser) 350, 361
  - Search Database (Browser) 372
- direct to SOM compiler options 168
  - get and set methods 169
  - implicit mode 168
  - SOM release order 170
- directories, for linker search 188
- disabling/enabling components 447
- DLL files
  - advantages of using 195
  - alignment factor in 202, 203
  - compiler option 155
  - exporting from
    - See* exporting from DLLs
  - include version string in 175
  - inserting description 537
  - linker option 207
  - linking to 194
  - linking to runtime 155
  - module statement for 540
  - naming 124
  - producing a 191
- /DLL linker option 207
  - load address of 203, 536

- DLLRNAME utility
  - /H option (help) 546
  - /N option (don't rename) 546
  - /Q option 546
  - /R option 547
  - command-line help 546
  - disabling logo and copyright statement 546
  - example 547
  - generating reports 547
  - how it works 545
  - options 546—547
  - syntax 544
- DLLs, tracing 434
- documentation, showing (Browser) 377
- double slashes as comments 148
- double-byte character set (DBCS) 147
- double-click actions (Browser) 346, 357
- DPATH environment variable 66
- DTS compiler options 168
  - get and set methods 169
  - implicit mode 168
  - SOM release order 170
- Dynamic Call Graph 485
  - menu bar 489
  - nodes and arcs 487
  - pop-up menus 492
  - status area 492
  - zoom bar 496
- dynamic link libraries, tracing 434
- dynamic linking 193
  - See also* DLL files
  - advantages of 195
  - description 92

## E

- E option for NMAKE 713
- echoing contents of linker response file 210
- edit boxes 636
  - properties, setting 648
- edit function 482
  - Call Nesting 482
  - Dynamic Call Graph 491
  - Execution Density 502
  - Statistics 508
  - Time Line 514
- edit menu, Trace Generation window 465
  - Call Nesting diagram 480
  - Execution Density diagram 501
  - Time Line diagram 512

- editing
  - resources 620, 623, 626, 691
    - dialog boxes 631—632
    - graphics 673
    - in graphic files 628
    - in script files 621
    - menus 657
    - string tables 669
- editing (Browser) 375
- editor 219
  - closing views 225
  - creating new files 219
  - customizing 249
  - editing text 220
  - entering text 220
  - finding marks 230
  - finding text 226
  - finding, replacing text 227
  - fonts 256
  - highlighting text 235
  - inserting files 232
  - introduction 219
  - Issue edit command window, using 233
  - issuing commands from 233
  - issuing multiple commands from 234
  - locating lines 228
  - manipulating blocks 235
  - marking text 235
  - marks 229
  - modifying behavior permanently 258
  - more on issuing commands from 234
  - multiple documents 239
  - naming marks 229
  - opening files 225
  - parsing 246
  - profiles 258
  - rings 239
  - saving files 224
  - special keys 220
  - tab settings 255
  - token attributes 257
  - tool bar, using 249
  - undoing changes 223
  - unhighlighting text 236
  - using quick marks 231
  - views 239
- EH\_CODE segment
  - renaming 162
- EH\_DATA segment
  - renaming 162
- enabling menu commands 661
- enabling threads 288
- enabling/disabling components 447
- ending the debugging session 270
- enlarging diagrams 476
- entry points, PERF 436
  - PerfStart and PerfStop 436
- enum variables, controlling size of 149
- enumerated variable 323
- environment variables 266
  - compiler 66
  - DPATH 66
  - HELP 66
  - ICC 66
  - ILINK 66
  - INCLUDE 66
  - LIB 66
  - LOCPATH 66
  - PATH 67
  - TMP 67
- error codes, compiler 80
- errors, compiler
  - controlling level and number 80
  - intermediate code linker 84
  - set maximum number of 135
- errors, linker
  - producing executable with 209
- escape characters for NMAKE 733
- event 445
- exception-handling information, removing 160
- exception-handling segments
  - renaming 162
- EXE files 78
  - adding stub file 542
  - alignment factor in 202, 203
  - compiler option 155
  - include version string in 175
  - inserting description 537
  - linker option 207
  - linking with errors 209
  - module statement for 540
  - naming 124, 212, 540
  - producing 207
  - producing an 190
  - running under DOS 542
  - search path 67
  - specify type of 212
  - subsystem 78
- /EXEC linker option 207

- Executable Information window 495
- executable pop-up menu 468
- executables, tracing 445
  - enabling/disabling 447
  - viewing/hiding 447
- executing a program 273
- Execution Density diagram 499
  - current column indicator 504
  - menu bar 500
  - pop-up menus 503
  - status area 503
  - vertical ruler 504
- execution time 487
- exiting Performance Analyzer 443
- exiting the Browser 336
- exported symbols, demangling 762
- exporting from DLLs
  - module statement 537
- EXPORTS module statement 537
- expression operands 323
- expression operators 324
- expression, monitor 300
- expressions supported 323
- EXTDEF symbols, demangling 761
- /EXTDICTIONARY linker option 208
- /EXTDICTIONARY option for ILIB 531
- Extended language level 73
  - compiler option 144
- external names
  - setting significant length of 172
- EXTMAKE syntax for NMAKE 736
- /EXTRACT object for ILIB 527

## F

- F option for LOCALDEF 610
- F option for NMAKE 713
- /Fa compiler option 122
- \_Far16** calling convention
  - keyword 90
  - setting 90
- fast floating-point execution 156
- fast integer execution 157
- fast-path keys (Browser) 415
- \_Fastcall** calling convention
  - keyword 90
  - setting 90
- Fb comp 335
- /Fb compiler option 78, 123

- /Fc compiler option 123
- /Fe compiler option 124
- /Fi compiler option 81, 94, 124
- file extensions
  - compiler output files 121
  - default 64
  - linker default 189
  - setting default for compiler 144
- file objects (Browser) 344
- files
  - as compiler option parameters 110
  - compiler options 121
  - editing (Browser) 375
  - executable (.EXE) files 78
  - extensions 64
  - help and message
    - See* message and help files
  - linker search rules 187
  - make
    - See* make files
  - memory 149
  - object 76
  - preprocessor 152
  - set default language of 150
  - setting default extension for compiler 144
  - source
    - See* compiler source files
  - viewing (Browser) 375
- filtering 475
- find function window 278
- finding text (Browser) 371
- /Fl compiler option 125
- flags, Browser generated 344
- floating menus 664
- floating-point execution, fast 156
- /Fm compiler option 125
- .FNT files 627
- /Fo compiler option 126
- fo resource compiler option 615
- fonts (Browser) 366
- /FORCE linker option 209
- /Fr compiler option 170
- frames 636
  - properties, setting 652
- /FREEFORMAT option for ILIB 530
- frequently used features 271
- /Fs compiler option 170
- /Ft compiler option 126
- function breakpoint 281

- function calls 445
- Function Information window 493
- function objects (Browser) 344
- function pop-up menu 469
- functions, enabling/disabling 447
  - viewing/hiding 447
- /Fw compiler option 82, 127
- FWAIT instruction 159

## G

- /Ga compiler option 168
- /Gb compiler option 169
- /Gd compiler option 92, 155
- /Ge compiler option 155
- /GENDEF (/gd) option for ILIB 530
- generated flags, Browser 344
- generating Browser databases 335, 336
- /GENIMPLIB (/gi) option for ILIB 531
- /Gf compiler option 156
- /Gh compiler option 77, 156
- /Gi compiler option 157
- /Gl compiler option 76, 157
- /Gm compiler option 93, 158
- /Gn compiler option 158
- graph overview (Browser) 354
- Graph window (Browser) 329, 351
  - action status bar 352
  - components 352
  - double-click actions 357
  - fonts 366
  - hold checkbox 352
  - information bar 353
  - initial action 357
  - layout 354
  - limitation 353
  - overview 354
  - printing 361
  - saving 361
  - selecting a zone 355
  - settings 356
    - bitmap dimensions 360
    - color 358
    - settings 356
    - style 359
  - slider 353
  - weighting 355
- Graphic editor (resources) 673
  - Airbrush tool 676
  - Colors palette 673

## Graphic editor (resources) *(continued)*

- copying and moving areas 680
- customizing 686, 687
- Eraser tool 678
- Hand tool 680
- hiding 687
- Line tool 677
- manipulating areas 679
- Paint Can tool 677
- Paint Pattern tool 676
- Paintbrush tool 676
- Pen Style tool 675
- Pick Rectangle tool 679
- Scissors tool 679
- selecting areas 679
- starting 673
- Text tool 678
- Tools palette 673
- window, splitting 686
- Zoom tool 680
- graphic files, resources 627—628
- graphics 673
  - adding text 678
  - aligning 679
  - brush shapes and patterns 676
  - colors 674
  - drawing and painting 676
  - editing scripts 673
  - enabling grids 688
  - erasing 678
  - flood-filling algorithm 678
  - large 680
  - moving 680
  - pen styles 675
  - RLE compression algorithm 682
  - source code for 688
  - testing 681
  - viewing 681
  - zooming 680
- group boxes 636
- /Gs compiler option 104, 158
- /Gu compiler option 84, 159
- /Gw compiler option 159
- /Gx compiler option 160
- /Gz compiler option 169

## H

- /H compiler option 172



- h message compiler option 739
- H option for CPPFILT 755, 759
- H option for DLLR 546
- H option for NMAKE 714
- h resource compiler option 616
- halt 287
- hardware requirements 263
- header files
  - controlling search paths 69
  - default file extensions 64
  - different calling conventions 160
  - precompiled 94
    - See also* precompiled header files
  - resources 626, 627
  - search path 66, 70
  - syntax 68
- heap check 287
- /HEAP linker option 209
- heap, defining size of 539
- HEAPSIZE module statement 539
- help
  - WorkFrame 7
- help and message files
  - search path 66
- help compiler (IPFC) 751
- HELP environment variable 66
- help from the command line 60, 171
- help levels (Browser) 365
- /HELP linker option 209
- help on the Browser
  - contextual 331
  - How Do I... 332
  - new user help (Browser) 366
- /HELP option for ILIB 531
- Help push button 463, 473
- help subtable resource 703
  - creating 705
  - editing 706
  - example 705
- help table resource 703
  - creating 705
  - editing 706
  - example 704
- HELPSUBTABLE statement 704
  - example 705
  - syntax 704
- HELPTABLE statement 703
  - example 704
  - syntax 703

- hide debugger on run 287
- history (Browser) 373
- hold checkbox (Browser) 341, 352
- hot spots 505
- hover help 11

## I

- /I compiler option 69, 70, 128
- I option for LOCALDEF 611
- I option for NMAKE 714
- i resource compiler option 616
- ibmpcni 764
  - ibmpcni 764
  - ibmpcnv 764
- ibmpcnv 764
- ibrs command
  - syntax 333
- icc command 59, 60
  - online listing of options 60
  - response files 61
  - syntax 60
- ICC environment variable 66
  - filenames in 67
  - include files in 70
  - include search path 69
  - precedence with command line 108, 112
  - setting compiler options in 107
- .ICO files 627
- icons (graphics resources) 636, 673, 689
  - colors, setting 683
  - creating 681
  - inverted areas 675
  - properties, setting 651, 683
  - removing images 681
  - testing 681
  - transparent areas 675
  - viewing 681
- icons, plus/minus 447
  - component 448
  - Trace Generation window 448
  - using to identify functions with triggers 449
- ICONV utility
  - creating conversion tables with
    - ICONVDEF 607
  - return codes 606
  - syntax 606
- ICONVDEF utility
  - return codes 607
  - syntax 607

- identifiers, resources 625—627
- /IGNORECASE linker option 215
- ignoring obsolete keywords 147
- ILIB
  - input 523
  - invoking 519
    - examples 522
    - using response files 521
    - using the command line 520
    - using the ILIB environment variable 521
  - objects 525
    - adding or replacing 526
    - extracting 527
    - removing 527
  - options 528—532
  - output 523
    - examples 524
  - specifying parameters 519
- ILINK environment variable 66
  - setting 198
  - setting options in 198
- implicit SOM mode 168
- import library, using
  - linker option 208
- improving compile time 94
- #include** directive 68
  - search path, controlling 69
  - syntax 68
- INCLUDE environment variable 66, 70
- #include** files
  - See also* header files
  - controlling 70
  - I option 128
  - in listing file, expanding user 133
  - in listing file, expanding user and system 133
  - search options 128
  - search order 70
  - system **#include** files 70, 71
  - user **#include** files 70, 71
  - Xc option 129
  - Xi option 129
- /INCLUDE linker option 209
- #include** search options 69, 70, 128
- incremental load (Browser) 336
- INF files, viewing 60
- inference rules for NMAKE
  - defining 725
  - example 726
  - predefined 727
- inference rules for NMAKE (*continued*)
  - setting file search path 726
  - TOOLS.INI 737
- information bar (Browser) 341, 353, 365
- /INFORMATION linker option 210
- initial sequence, precompiled headers
  - customizing in source files 100
  - determining 95
  - matching 96
  - multiple 98
  - reusing 96
- inline files for NMAKE 732
- \_Inline** keyword 85
- inlining user code 85
  - benefits 88
  - compiler option 86, 163
  - description 85
  - drawbacks 89
  - improving performance by 88
  - keywords 85
  - limit working set size 164
  - restrictions 89
- instruction scheduler, invoking 165
- integer execution, fast 157
- intermediate code linker
  - compiler option 164
  - description 81
  - error checking 84
  - Gu option 84
  - restrictions 83
  - with Gu option 159
- intermediate files 82
  - directing 127
  - linking 81
  - naming 127
  - naming in ICC 67
  - producing 127
- internationalization utilities
  - ICONV 605
  - ICONVDEF 607
  - LOCALDEF utility 609
- intrinsic functions 86
- introducing the basic windows 305
- introduction to the debugger 263
- invoking the instruction scheduler 165
- IPF compiler (IPFC)
  - syntax 751
- IRC (resource compiler) 615, 627
- isolating problems 449

## J

/J compiler option 143  
jump to location choice 302

## K

keyboard customization 252  
keywords, ignoring obsolete 147

## L

/L compiler option 131  
/La compiler option 131  
label objects (Browser) 344  
language level  
    ANSI 72  
    compiler option 144  
    description 71  
    extended 73  
    SAA 72  
    specifying 71  
language-dependent compiler options 113  
/Le compiler option 132  
/Lf compiler option 132  
/Li compiler option 133  
LIB environment variable 66  
LIB files  
    linker search rules 187  
    linking with 194  
    naming in ICC 67  
libraries, Performance Analyzer 434  
libraries, specifying  
    default 91  
    for subsystem EXE files 78  
    in ICC 67  
    multithread 93, 158  
    naming conventions 92  
    runtime 91  
    search path 66  
    single thread 94  
    suppressing default library information 158  
    using compiler options 115  
    with different calling conventions 160  
library DLLs, packaging with your  
    application 543  
library files  
    linking with 193  
    naming conventions 92  
    searching default 206

library manager

*See also* ILIB

    introduction 519

LIBRARY module statement 540

limitation (Browser) 341, 353

limitations 268

line breakpoint 279

line breakpoint window 279

/LINENUMBERS linker option 210

linkage

*See also* calling conventions

    keywords 90

linker 177

    case sensitivity 215

    command-line syntax 180

    default library 91

    default options 182

    dynamic 92

    errors, overriding 209

    filename defaults 189

    HEAP storage 209

    input 187

    logo display 210

    map files

*See* map files

    module statements 536

    naming output 212

    numeric arguments 199

    object files 189

    optimizations 185

    option for debugger and Performance

        Analyzer 77

    options 201

    output 187

    passing options from compiler 172

    response files 181

    return codes 192

    search rules 187

    searching additional directories 188

    source file line numbers in 210

    specifying executable type 190

    starting the 179

    static 92

linker errors

    producing executable with 209

linker options 201, 202

    in the ILINK environment variable 198

    invalid 181

    numeric arguments 199

    on the command line 197

- linker options (*continued*)
  - setting 197
- linker options in WorkFrame 198
- linking
  - a DLL file 191
  - an .exe file 190
  - description 187
  - displaying status of 210
  - echoing contents of response file 210
  - for the debugger 263
  - from a makefile 183
  - from the command line 180
  - ignoring capitalization 215
  - ignoring errors 209
  - in WorkFrame 179
  - managing HEAP storage 209
  - object files 189
  - referencing a symbol 209
  - specifying executable type 190
  - through the compiler 182
  - to DLLs 194
  - to runtime libraries 155
  - using extended dictionary 208
  - using response files 181
  - with LIB files 194
  - without default library information 158, 206
- linking (Browser) 335
- linking programs 433
- linking resources to applications 627
- list boxes 636
  - properties, setting 650
- list breakpoints 286
- /LIST option for ILIB 531
- List window (Browser) 329, 340
  - action status bar 341
  - components 341
  - container view 341
  - double-click actions 346
  - fonts 366
  - hold checkbox 341
  - information bar 341
  - initial action 346
  - limitation 341
  - ordering 342
  - printing 350
  - saving 350
  - settings 345
    - color 347
    - settings 346
    - style 349
- List window (Browser) (*continued*)
  - text attributes 347, 349
  - types of 341
- listing file compiler compiler options 130
- listing files, compiler
  - assembler 75
  - compiler options 130
  - cross-reference table in 135
  - description 79
  - directing compiler 125
  - expand macros in 132
  - expand user **#include** files in 133
  - expand user and system **#include** files
    - in 133
  - include source code 134
  - information about variables in 130
  - maximum information in 132
  - minimum contents 79
  - naming compiler 125
  - producing 131
  - producing compiler 125
  - referenced struct and union variables in 131
  - set page length of 133
  - set subtitle string in 134
  - set title string of 134
  - source program information in 130
- /Lj compiler option 133
- load address 203, 536
- load occurrence breakpoint 285
- load segment 203, 536
- load segments 203, 536
- load-on-call DLLs, tracing 434
- loading files (Browser) 368
- Local Variable Window 313
- LOCALDEF utility
  - controlling messages 611
  - locale source file 611
  - options 610—611
  - return codes 611
  - specifying character map file 610
  - syntax 609
- locale files
  - LOCALDEF utility 609
  - specifying source file for LOCALDEF 611
- locate text (Browser) 371
- location, jump to 302
- LOCPATH environment variable 66
- /LOGO linker option 210
- /LOGO option for ILIB 532

- logo, disabling
  - compiler 102
  - linker 210
- /Lp compiler option 133
- /Ls compiler option 134
- /Lt compiler option 134
- /Lu compiler option 134
- /Lx compiler option 135
- /Ly compiler option 135

**M**

- /M compiler option 90, 160
- M option for CPPFILT 755
- machine-state dump 138
- macros
  - NMAKE
    - command line 719
    - defining 718
    - inherited 720
    - predefined 721
    - substitutions 721
    - using 720
- magnifying diagrams 476
- main control window 458
- make file utility (NMAKE) 709
- make files
  - compiling from 62
  - linker 183
- MakeMake utility 52
  - command line syntax 55
  - using 52
  - window 52
- managing libraries (.LIB files) 519
- mangled names, removing underscore 762
- map files
  - compiler option 125
  - linker option 211
  - producing 191
  - source file line numbers in 210
- /MAP linker option 211
- maximum execution time 488
- MAXVAL attribute 539
- memory files, allowing use of 149
- memory-protection attributes of segments 213
- menu bar, customizing menus 251
- menu descriptions (Browser) 415
  - popup menus 424
    - for objects 425
    - for windows 424

- menu descriptions (Browser) (*continued*)
  - pulldown menus 416
    - Actions 420
    - Edit 418
    - File 417
    - Help 423
    - Options 421
    - Order 422
    - Project 423
    - View 419
    - Windows 422
- Menu editor (resources) 657
  - customizing 664
  - starting 657
- menu resource 657
  - adding to menu bars 658
  - associating with shortcuts 659
  - deleting 663
  - editing scripts 657
  - floating 664
  - moving 663
  - separators 659
  - source code for 668
  - testing 664
  - viewing 657
- menuex resource 657
  - editor 665
- menus
  - dialog boxes 633
- merging files (Browser) 369
- message and help files
  - search path 66
- message compiler (MC)
  - command line syntax 739
  - formatting a message string 746
  - input message file 740
    - example 745
    - header section 740
    - language-specific message text
      - definitions 743
    - message definitions 742
  - introduction 739
  - options 739—740
- messages
  - compiler options 80
  - controlling 138
  - setting severity level of compiler 138
- migration
  - language standards for C++ 73

- miscellaneous compiler options 171
- modifying editor behavior 258
- module definition files
  - example 534
  - linker search rules 187
  - module statements 536
  - naming in ICC 67
  - numeric arguments 199
  - reserved words 534
  - rules 533
  - when to use 533
- module statements 536
- monitor expression 300
- Monitor Windows 316
- moving controls (resources) 638
- multiple controls, selecting (resources) 637
- multiple views 477
- multithread
  - library 93

## N

- /N compiler option 135
- n instances 345
- N option for CPPFILT 759
- N option for DLLR 546
- N option for NMAKE 714
- NAME module statement 540
- naming compiler output 124
- naming conventions for libraries 92
- /Nd compiler option 161
- new user help (Browser) 366
- NMAKE utility
  - building all targets (/A option) 713
  - description files
    - creating 727
    - description blocks 715
    - format 716
  - directives 727—729
  - displaying help 711, 714
  - displaying modification dates (/D option) 713
  - escape characters 733
  - example of directives 730
  - example of inference rules 726
  - example of inline files 733
  - EXTMAKE syntax 736
  - ignoring errors (/I option) 714
  - ignoring inference rules (/R option) 715
  - inference rules 725
  - NMAKE utility (*continued*)
    - inline files 732
    - invoking 709
    - macros
      - command line 719
      - defining 718
      - example of predefined 722
      - inherited 720
      - modifiers 724
      - precedence rules 724
      - predefined 721
      - substitutions 721
      - using 720
    - modifying commands 734
    - options 712—715
    - overriding environment variables (/E option) 713
    - predefined inference rules 727
    - printing definitions (/P option) 714
    - producing error file 712
    - pseudotargets 730
    - removing error checking (- command) 734
    - response files 711
    - running commands for dependents (! command) 735
    - running from batch file 714
    - showing commands without running (/N option) 714
    - specifying description files (/F option) 713
    - specifying file names 723
    - specifying search directories 716
    - suppressing command echo 735
    - suppressing command echo (/S option) 715
    - suppressing logo display 714
    - suppressing messages (/C option) 713
    - syntax for command line 710
    - target in multiple description blocks 717
    - TOOLS.INI 737
  - /NOBACKUP option for ILIB 529
  - /NOBASE linker option 203
  - /NOBROWSE linker option 204
  - /NOBROWSE option for ILIB 530
  - /NODBGPACK linker option 205
  - /NODEBUG linker option 206
  - /NODEFAULTLIB linker option 206
  - nodes, Dynamic Call Graph 487, 488
    - color 487
  - /NOEXTDICTIONARY linker option 208
  - /NOEXTDICTIONARY option for ILIB 531

- /NOFREEFORMAT option for ILIB 530
- /NOIGNORECASE linker option 215
- /NOINFORMATION linker option 210
- /NOLINENUMBERS linker option 210
- /NOLOGO linker option 210
- /NOLOGO option for ILIB 532
- NOLOGO option for NMAKE 714
- /NOMAP linker option 211
- NONAME keyword, CPPFILT-generated 759
- /NOPTFUNC linker option 211
- /NOQUIET option for ILIB 532
- notebooks
  - Browser Settings 363
  - Graph window Settings (Browser) 356
  - List window Settings (Browser) 345
- /NOWARN option for ILIB 532
- /Nt compiler option 161
- numeric arguments for the linker 199
- numsign line directive 153
- /Nx compiler option 162

## O

- /O compiler option 76, 162
- O option for CPPFILT 760
- object (.OBJ) files
  - creating 76
  - directing 126
  - formats acceptable for linker 189
  - include version string in 175
  - linker search rules 187
  - naming 126
  - naming in ICC 67
  - producing 126
  - specifying for linker 189
- object (Browser) 425
- object file pop-up menu 468
- object files, enabling/disabling 447
  - viewing/hiding 447
- object-action pair 331
- objects (Browser) 356
- objects, Browser 330, 343
- obsolete keywords, ignoring 147
- /Oc compiler option 76, 163
- ODBC
  - configuring data sources 582
  - DB2 driver 587
  - driver names 582
  - drivers 581
  - error messages 582

- ODBC (*continued*)
  - getting started 581
  - installing 582
  - isolation levels 584
  - locking 584
  - Oracle 7 driver 588
  - supported database systems 581
  - Sybase System 10 driver 593
  - Windows 3.x run-time support 582
- /Oi compiler option 76, 86, 163
- OK push button 463, 473
- /Ol compiler option 76, 82, 164
- /Om compiler option 164
- online information, creating with IPF 751
- /Op compiler option 76, 164
- Open Database Connectivity
  - See* ODBC
- open new source debugger choice 277
- open new source window 277
- operands, valid expression 323
- operators, valid expression 324
- /OPTFUNC linker option 185, 211
- optimizing
  - compiler options 162, 163
  - description 76
  - DLL files 191
  - EXE files 190
  - for size and speed 163
  - for speed 162
  - inlining user code 85
  - instruction scheduler 165
  - intermediate code linker 81
  - linking 185
  - removing unreachable functions 185
  - removing unreferenced functions 157, 211
  - use of stack pointers 164
  - with debugger 77
  - with intermediate linking 159
- options menu, Window Manager window 459
  - Call Nesting diagram 481
  - Dynamic Call Graph 490
  - Execution Density 501
  - Statistics diagram 507
  - Time Line diagram 513
  - Trace Generation window 466
- options, compiling and linking 433
- \_Optlink** calling convention
  - compiler option 160
  - keyword 90
  - setting 90

Oracle

*See* ODBC

ordering a container view (Browser) 342

ordinal position of data construct 538

ordinal position of function 538

ordinals, generating with CPPFILT 760

organizing a graph (Browser) 354

organizing source files for precompiled headers 100

/Os compiler option 76, 165

other compiler options 171

/OUT linker option 212

/OUT option for ILIB 532

output from compiler 75

output options, compiler 121

overhead time 438

overview, graph (Browser) 354

## P

/P compiler option 152

P option for CPPFILT 760

P option for NMAKE 714

parameters for ILIB 519

parameters of compiler options 109

    files 110

    numbers 110

    strings 109

    switches 110

parts

*See* project parts

PATH environment variable 67

paths (Browser) 363

patterns, recognizing 477

/Pc compiler option 152

/Pd compiler option 152

PDB, PDD 333, 336, 337, 368, 370

/Pe compiler option 153

PERF entry point 435

Performance Analyzer

    compiler option 136, 156

Performance Analyzer - Specify Profile Location

    window 457

Performance Analyzer - Window Manager

    window 458

        menu bar 459

        pop-up menus 460

        push buttons 461

Performance Analyzer, description 431

    exiting 443

Performance Analyzer, description (*continued*)

    PERF entry point 436

    PerfStart and PerfStop entry points 436

    starting from a command line 439

    starting from WorkFrame 441

performance, improving executable 88

PerfStart and PerfStop entry points 436

PM programming, compiler options for 114

/PMTYPE linker option 212

pop-up windows 631

    dialog boxes 631

    frames 652

    menus 633

    overlapping 635

    properties, setting 632

popup menu, thread 275

popup menus (Browser) 424

    for objects 425

    for windows 424

portability

    language standards 72

    publications 799

#pragma directives

    alloc\_text 161

    dataseg 161

    hdrfile 94

    hdrstop 95

    langlvl 72

    linkage 90

precedence of compiler options 108

precompiled header files

    default names 98

    description 81

    determining initial sequence 95

    examples 98

    Fi option 124

    improving compile time with 100

    matching the initial sequence 96

    multiple initial sequences 98

    organizing source files for 100

    restrictions 81

    reusing 96

    Si option 146

    strategies 100

    using 94

preparation tasks 335

preprocessor

    compiler option 152

    directing output 152

    directives 68



- preprocessor (*continued*)
  - files, creating 152
  - generation of #line directives 153
  - macros, defining 151
  - macros, undefining 153
  - options 151
  - run only the 152
  - using the 151
- preprocessor compiler options 151
- preprocessor files
  - #line directives in 153
  - creating 152
  - including comments in 152
  - writing to stdout 152
- printing
  - graphs (Browser) 361
  - lists (Browser) 350
- Process List window 268
- processes, attach 268
- processes, detach 268
- producing linker map files 191
  - compiler option 125
  - linker option 211
  - with source file line numbers 210
- profile hooks 433
- profile, specifying 457
- profiles, using to modify editor behavior 258
- profiling hooks 78
- program file, Browser 333, 337
- program startup window 265
- programs, preparing 433
  - tracing 445
- project parts 9
  - filtering 13
  - working directory 9
    - specifying in project settings 28
- Project Smarts 16, 59
  - creating projects from 17
  - location page 17
  - project page 17
  - starting from command line 17
  - using 17
- projects 9
  - build actions 40
  - creating 15
    - by copying 15
    - using Project Smarts 15, 16
  - dependencies between 19, 37
  - environment variables 30
  - filtering parts 13
- projects (*continued*)
  - generating makefiles for 52
  - geometry 23
  - hover help 11
  - information line 13
  - introducing 9
  - makefile 26
    - maintaining 37
    - specifying 26
  - menus 12
  - name 31
  - nesting 19
  - options dialog 32
  - options file 9
  - organizing for builds 19
  - parts 9
  - parts container 13
  - parts filter 13
  - project file 9
  - Project Smarts 16
  - run options 26
  - Settings notebook 25
  - source directories 9
    - displaying in tree view 14
    - organizing projects 23
    - specifying in project settings 28
    - storing nested projects 20
    - storing source files in 22
  - structures 23
  - subprojects 19
  - target 26
  - toolbar 11
  - tools options setup 32
  - tools setup 9
  - views 10
    - icon view 10
    - tree view 14
- properties
  - dialog boxes 632—635
- prototypes, for the PERF entry point 435
  - for the PerfStart and PerfStop entry points 436
- pseudotargets 730
  - .IGNORE 731
  - .PRECIOUS 731
  - .SILENT 731
  - .SUFFIXES 731
  - predefined 731
- public symbols, demangling (PUBDEF, COMDAT, COMDEF) 760

- publications
  - related 799
- pulldown menus (Browser) 416
  - Actions 420
  - Edit 418
  - File 417
  - Help 423
  - Options 421
  - Order 422
  - Project 423
  - View 419
  - Windows 422
- push button, trace 470
- push buttons 636
  - properties, setting 646

## Q

- /Q compiler option 102, 173
- Q option for CPPFILT 755, 761
- Q option for DLLRNAME 546
- Q option for NMAKE 714
- /qalias compiler option 165
- /qautoimported compiler option 173
- /qautothread compiler option 173
- /qbitfields compiler option 143
- /qdbgunref compiler option 136
- /qdigraph compiler option 143
- /qignprag compiler option 174
- /qisolated\_call compiler option 165
- /qlibansi compiler option 174
- /qlonglong compiler option 144
- /qmakedep compiler option 174
- /qro compiler option 175
- /qsomvolattr compiler option 175
- /qtune compiler option 166
- QuickBrowse 329, 389
- /QUIET option for ILIB 532
- /qwin32s compiler option 166

## R

- /R compiler option 78, 94, 167
- r message compiler option 739
- R option for CPPFILT 761
- R option for DLLRNAME 547
- R option for NMAKE 715
- radio buttons 636
  - properties, setting 647

- rectangle control 636
  - properties, setting 652
- rectangular block of text, marking 237
- redo actions (Browser) 373
- reducing diagrams 476
- reducing trace data 447, 475
- Register Window 308
- related compiler options 112
- related publications
  - portability 799
  - VisualAge for C++ 799
- relocating files 208
- /REMOVE object for ILIB 527
- removing exception-handling information 160
- removing stack probes 104
- removing unreferenced functions 157, 211
- renaming DLLs (DLLRNAME) 544
- resizing controls (resources) 638
- resizing window panes 506
- resolving templates 126
- resource compiler (IRC) 615, 627
  - options 615
- Resource Image Conversion Utility
  - command line processor, conversion utility 765
- resource linker 627
- resource script files 620—627
  - adding scripts 621
  - compiling and linking 627
  - creating 620
  - editing 621
  - identifiers 625—627
- resource scripts 620
  - copying and moving 623
  - creating new 621, 622
  - customizing resources 691
  - deleting 625, 626
  - editing 623, 626
  - entering data 699
  - specifying a language version 624
  - specifying memory options 624
- Resource Tools
  - overview 615
  - resource compiler 615
  - resource linker 616
- Resource Workshop
  - custom data types 691, 698, 701
  - customizing 628
  - editors 620, 623
    - accelerators 665
    - custom data types 691

- Resource Workshop (*continued*)
  - editors (*continued*)
    - customizing 664, 668, 671, 686
    - dialog boxes 631
    - graphics 673
    - help subtables 706
    - help tables 706
    - menuex 665
    - menus 657
    - scripts 691
  - error messages 669, 700
  - overview 617
  - starting 619
- response files
  - compiler 61
  - linker, echoing contents of 210
  - using linker 181
- response files for ILIB 521
- response files for NMAKE 711
- return codes, compiler 80
- return codes, linker 192
- returns, in Call Nesting 477
- run 287
- runtime libraries, choosing 91
- runtime libraries, linking to 155
- runtime library DLLs for VisualAge for C++
  - packaging with your application 543

## S

- /S compiler option 72, 73, 144
- s message compiler option 739
- S option for CPPFILT 756, 761
- S option for NMAKE 715
- SAA Level 2 language level 72
  - compiler option 144
- saving
  - graphs (Browser) 361
  - lists (Browser) 350
- scaling 476
  - Execution Density 504
  - Time Line 516
- Script editor (resources) 691
  - starting 691
- scrolling 476
- /Sd compiler option 144
- search order for header files 70
- search order of header files 70
- search path
  - controlling **#include** 69

- search path (*continued*)
  - EXE files 67
  - help files 66
  - INCLUDE environment variable 66
  - include, controlling 69
  - libraries 66
  - message and help files 66
- search rules for linker 187
- searching (Browser) 372
- searching extended dictionary 208
- /SECTION linker option 213
- segments
  - constant
    - See* constant segment
  - default attributes 213
  - default code and text
    - See* default code and text segments
  - default data
    - See* default data segment
  - DGROUP 161, 539
  - exception-handling
    - See* exception-handling segments
  - load 203, 536
    - See also* load segments
  - maximum number of 213
  - renaming 161, 162
  - setting attributes of 213
  - uninitialized data
    - See* uninitialized data segment
- /SEGMENTS linker option 213
- selecting
  - bitmap dimensions (Browser) 360
  - graph zone (Browser) 355
- sequence numbers in compiler source files 148
- set address breakpoint 283
- set change address breakpoint 283
- set function breakpoint 281
- set line breakpoint 279
- set load occurrence breakpoint 285
- set maximum number of compiler errors 135
- setlocale( ) function 66
- setting breakpoints 274, 275
- setting compiler options
  - in ICC 107
  - in WorkFrame 108
  - on the command line 107
- setting line breakpoint 279
- setting stack size 105
- setting up projects and tool options 25

- settings 25
  - Browser 363
  - build 39
    - actions page 40
    - display page 46
    - make page 42
    - project page 44
  - Graph window (Browser) 356
    - bitmap dimensions 360
    - color 358
    - settings 356
    - style 359
  - List window (Browser) 345
    - color 347
    - settings 346
    - style 349
  - project 25
    - directories page 28
    - environment page 30
    - name page 31
    - target page 26
- settings, saving trace file 453
  - profile 457
- severity of messages 138
- severity, compiler return codes 80
- /Sg compiler option 145
- /Sh compiler option 146
- shipping VisualAge for C++ DLLs with your product 544
- showing
  - documentation (Browser) 377
- /Si compiler option 81, 94, 146
- single thread
  - library 94
- size of enum variables, controlling 149
- size, node 488
- /Sm compiler option 147
- /Sn compiler option 147
- software requirements 263
- SOM compiler options 168
  - get and set methods 169
  - implicit mode 168
  - SOM release order 170
- sorting, Statistics 507
- source code compiler compiler options 143
- source files, compiler
  - compiling multiple 63
  - default file extensions 64
  - double slashes in C files 148
  - file types 64
- source files, compiler (*continued*)
  - ignoring columns in 145, 148
  - ignoring sequence numbers in 148
  - include source code in listing file 134
  - information about in listing 130
  - language level 71
  - line numbers in map file 210
  - mixed C and C++ 63
  - naming in ICC 67
  - options with multiple 111
  - organizing for precompiled headers 100
  - sequence numbers in 148
  - setting default extension 144
  - setting margins of 145
  - source code compiler options 143
  - specifying C files 149
  - specifying C++ files 150
- source windows 275, 293
- source, open new - debugger choice 277
- /Sp compiler option 147
- specifying libraries
  - default 91
  - for subsystem EXE files 78
  - in ICC 67
  - multithread 93, 158
  - naming conventions 92
  - runtime 91
  - search path 66
  - single thread 94
  - suppressing default library information 158
  - using compiler options 115
  - with different calling conventions 160
- /Sq compiler option 148
- /Sr compiler option 148
- /Ss compiler option 148
- stack
  - allocation of 102
  - default size 105
  - linker option 214
  - module statement 541
  - pointers, optimizations involving 164
  - restrictions on size 214
  - setting size of 105
- /STACK linker option 105, 214
- stack probes
  - compiler option 158
  - description 103
  - removing 104, 158
- STACKSIZE module statement 105, 541

- start tracing 471
- starting the Browser
  - from the command line 333
  - from the Debugger 334
  - from the Program Manager 334
  - from WorkFrame environment 334
- starting the compiler 59
  - using the system function 59
- starting the debugger from WorkFrame 269
- startup commands 439
- static linking 193
  - description 92
- static text controls 636
  - properties, setting 647
- Statistics diagram 505
  - Details pane 509
  - menu bar 506
  - Summary pane 509
- status area, Call Nesting 483
  - Dynamic Call Graph 492
  - Execution Density 503
  - Time Line 515
- \_\_stdcall** calling convention
  - compiler option 160
- step over 301
- stop push button 471
- stop tracing 471
- Storage Window 309
- String editor (resources) 669
  - customizing 671
  - starting 669
- string tables 669—671
  - adding to scripts 669
  - allocating memory 670
  - changing 670
- strings
  - as compiler option parameters 109
- structure and union table 130
- structures, aligning data items in 147
- stub file 542
  - linker search path 542
- STUB module statement 542
- /Su compiler option 149
- /SUBSYSTEM linker option 215
- subsystems
  - compiler option 167
  - thread support 94
- summary of compiler options 117
- Summary pane 509
- /Sv compiler option 149
- switches (+|-) for compiler options 110
- Sybase
  - See* ODBC
- symbol map, producing with CPPFILT 755
- system API calls, tracing 434
- \_System** calling convention
  - compiler option 160
  - keyword 90
  - setting 90
- system exceptions 512
- system include files 70, 71
- system object model compiler options 168
  - get and set methods 169
  - implicit mode 168
  - SOM release order 170

## T

- T option for CPPFILT 756
- T option for NMAKE 715
- /Tc compiler option 63, 149
- /Td compiler option 63, 150
- templates, resolving 126
- temporary files
  - description 79
  - storing on disk or in memory 79
- testing resources 653, 664, 681
- thread enabled 288
- thread interactions 479
- threads box 275
- threads, call depth 449
- /Ti compiler option 77, 136
- time data 449
- Time Line diagram 511
  - menu bar 512
  - pop-up menus 515
  - status area 515
  - vertical ruler 516
- time on stack 486
  - definition 489
  - with node size 487
- time stamps 449
- time-scaled diagrams 477
- time, overhead 438
- /Tm compiler option 137
- TMP environment variable 67
  - compile time 79
- /Tn compiler option 137

- token attributes
  - editor 257
- tool bar 271
- tool bar buttons 271
- TOOLS.INI 737
- touching modification dates with NMAKE 715
- tour (Browser) 395
- /Tp compiler option 63, 150
- trace data, reducing 447
- trace event 445
- trace file description 447
- trace files, steps before creating 433
  - analyzing, description 455
  - controlling the size of 447
  - creating 445
  - customizing 447
  - displaying in diagrams 472
  - naming 450
  - opening in diagrams 456
  - settings, saving 453
  - viewing 475
- Trace Generation window, enabling/disabling
  - components on 448
  - menu bar 464
  - pop-up menus 468
  - trace push button 470
- trace off push button 471
- trace on push button 471
- treatment of unspecified char 143
- trends 499
- triggers 449
- troubleshooting (Browser)
  - added menu items don't work 413
  - error loading PDB files 413
  - error loading program files 413
  - graph zone won't maximize 414
  - won't start 413
- /Tx compiler option 138
- type conversion 148
- type objects (Browser) 344
- typecasting, data types in monitors 325

**U**

- /U compiler option 153
- U option for CPPFILT 762
- undefining preprocessor macros 153
- uninitialized data segment
  - renaming 161
- unions, aligning data items in 147
- unreachable functions, removing 185
- unreferenced functions, removing 157, 211
- unspecified char, treatment of 143
- updating Browser database 392
- user events, creating 435
  - in Statistics 510
  - locating in Call Nesting 480
  - locating in Time Line 513
- user include files 70, 71
- user interface
  - Browser 339
- using an import library
  - linker option 208
- using the Browser
  - listing
    - all classes 378
    - all files 378
    - all friends 380
    - all friendships 381
    - all objects defined 379
    - callers and callees 382
    - implementing files 380
    - instantiations 385
    - members of a class 383
    - overriding derived classes 384
    - possible exceptions 385
  - viewing
    - call chains 387
    - class relationships 386
    - include file relationships 388
- using the Mouse (Browser) 331

**V**

- /V compiler option 175
- v message compiler option 739
- v resource compiler option 616
- variable objects (Browser) 344
- version information (resources) 691, 700
- version string in compiler output 175
- vertical ruler, Execution Density 504
  - Time Line 516
- view command 60
- view menu, Window Manager window 459
  - Call Nesting diagram 481
  - Dynamic Call Graph 490
  - Execution Density diagram 501
  - Statistics diagram 507
  - Time Line diagram 513

view menu, Window Manager window  
(*continued*)

Trace Generation window 466

viewing (Browser) 375

viewing INF files 60

views, multiple 477

views, project 10

icon view 10

tree view 14

VisualAge for C++

publications 799

## W

/W compiler option 138

-w message compiler option 739

W option for CPPFILT 756

W option for LOCALDEF 611

/WARN option for ILIB 532

/Wgrp compiler option 138

groups 140

where is execution point debugger choice 278

Who Calls Whom window 496

wildcards

in names of compiler source files 65

windows 457

Analyze Trace 472

Application Monitor 470

Create Trace 462

Graph (Browser) 329, 351

History (Browser) 373

List (Browser) 329, 340

Overview (Browser) 354

Performance Analyzer - Specify Profile

Location 457

Performance Analyzer - Window

Manager 458

resizing 506

Trace Generation 463

Windows exceptions

guard page 103

machine-state dump 138

work files 67

work files, compiler 79

WorkFrame 3

actions 9

build utility 38

Console window 59

getting help on 7

contextual 7

**How Do I?** 7

WorkFrame (*continued*)

hover help 11

initial dialog 4

linking in 179

makefile generation utility 37, 52

MakeMake utility 37, 52

monitor 5

options file 9

overview 3

project file 9

Project Smarts 59

projects

*See* projects

setting compiler options in 108

starting from command line 4

starting the Browser from 334

WorkFrame, linker options in 198

WorkFrame, with the Performance Analyzer 441

working set size of compiler 164

## X

X option for CPPFILT 762

X option for NMAKE 712

-x resource compiler option 616

/Xc compiler option 69, 70, 129

/Xi compiler option 69, 70, 129

/Xs compiler option 169

## Z

Z option for CPPFILT 762

zone, graph (Browser) 355

zoom bar 496

zooming 476

in Dynamic Call Graph 496

in Execution Density 503

in Time Line 513



Program Number: 33H4979

Printed in U.S.A.

S33H-5031-00

