



[Home](#)

Using Hawk™

Version 2.5



RadiSys.
THE POWER OF WE

www.radisys.com
Revision A • July 2006

Copyright and publication information

This manual reflects version 2.5 of Hawk.

Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microwave Communications Software Division, Inc.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

July 2006
Copyright ©2006 by RadiSys Corporation
All rights reserved.

EPC and RadiSys are registered trademarks of RadiSys Corporation. ASM, Brahma, DAI, DAQ, MultiPro, SAIB, Spirit, and ValuePro are trademarks of RadiSys Corporation.

DAVID, MAUI, OS-9, OS-9000, and SoftStax are registered trademarks of RadiSys Corporation. FasTrak, Hawk, and UpLink are trademarks of RadiSys Corporation.

† All other trademarks, registered trademarks, service marks, and trade names are the property of their respective owners.

Contents

Introduction 19

Hawk Overview	20
The Hawk Project Manager	20
The Hawk Editor	20
The Hawk Debugger	26
The Hawk Compiler	26

Running Hawk for the First Time 27

A First Look.....	28
The Menu Bar	28
File Menu	28
Edit Menu	29
Search Menu	30
Project Menu	30
Debug Menu	31
Target Menu	31
Text Menu	31
Document Menu	32
The Customize Menu	33
Tools Menu.....	34
Window Menu	35
Window Menu.....	36
Help Menu.....	36
Output and Project Windows.....	36
The Output Window.....	36
Auto-Increment Search Tabs	37
Append to Listing.....	37
The Project Window	37
The Standard Toolbar.....	40

Building Your First Project 41

Projects and Project Spaces.....	42
What are Hawk Projects and Project Spaces?	42
Creating a Project Space.....	42
Setting Project Defaults	42
Creating and Modifying a Project.....	43
Adding Files to a Project.....	45

Command Key, Libraries, and General Environment 49

API Command Dialog/Prompt.....	50
API Command Key/Prompt	50
Command Completion	51
Example	51
Loading Hawk Add-Ons	52



General Environment Settings 52

View Setups and Language Support 55

View Setups 56

 Hawk View Setups 57

 Default View Setup 57

 Output Window View Setup 58

 Using View Setups 58

Hawk Language Support 58

 Language Support Lexers and DLL's 59

 Language DLLs 59

 The Tools Dialog 60

 ChromaCoding Lexers 60

 Creating a Lexer 61

 Examples 61

 Configuring Options in the Language Dialog 67

 File Type List 68

 Options Tab 68

 Tabs/Indenting Tab 68

 Templates Tab 69

 Coloring Tab 69

 CodeSense Tab 71

 Format Tab 71

 Comments Tab 71

 Adding a New File Type to the Language Dialog 71

 Aliasing 72

Editing Features 73

 Templates and Brace Expansion 74

 Templates 74

 Language-Specific Templates 74

 Creating and Modifying Language-Specific Templates 74

 Examples 75

 Non-Language-Specific Templates, Function and File Headers, and Macros in Templates 75

 CodeFolio Snippets 79

 Using an Existing Code Snippet 80

 TITLE.TPL 80

 VBFUNCT.TPL 80

 Adding a Code Snippet 81

 Creating a Code Snippet from the Current Document or Clipboard 81

 Deleting or Renaming a Snippet 82

 Editing a Snippet 82

 Adding or Removing Snippets Directories 82

 Brace Matching and Brace Expansion 83

 Brace Function: Finding Unmatched Braces 83

 Command: Brace TRUE font 84

 Brace Highlighting 84

 Brace Locating 84

 BraceFindEx() 84

 Example 85

 Brace Expansion 85



Align Beginning and End of Block	85
Example	85
Indenting.....	86
Setting Spaces and Tabs	86
Example	86
Seek Indentation and Smart Indenting.....	86
Example	87
Block Alignment.....	87
Name Completion	88
CodeSense.....	88
Where CodeSense Gets its Information.....	88
Library and Project Databases	88
Add a File to Symbols Window File List	89
CodeSense for Files that are Open in Hawk.....	90
CodeSense Global Configuration Dialog.....	90
Create CodeSense Library Database.....	91
Edit CodeSense Library Database Location	92
Delete CodeSense Library Database.....	92
Parser Priority/Resource Use.....	93
CodeSense Databases.....	93
Database Files.....	93
Database Corruption.....	94
Using CodeSense	95
Name Completion.....	95
Auto-list Members (Standard)	96
Auto-list Members (Special Types).....	96
Auto-type Info.....	96
Auto-Parameter Info.....	96
Comments and Comment Boxes.....	97
HTML Editing.....	98
HTML Language Support.....	98
HTML Popup Menu.....	98
HTML Viewing – HTML WYSIWYG Editor/Viewer and Web Browser Interface.....	98
Viewing Internet Files: Installation Instructions.....	99
HTML Toolbar	99
Using HTML WYSIWYG.....	99
Hex Editing.....	101
Insert vs. Overtyping Mode	101
Handy Hawk Hex-Editing Features	101
Clipboard and Scrap Buffers.....	102
Scrap Buffers	102
Multiple Clipboard/Scrap Buffers.....	102
Clipboard/Scrap Viewer.....	104
API Assistant	105
Using the API Assistant.....	105
Using the Checkboxes	105
API Assistant Example.....	105
Modifying the Database.....	106
API Databases.....	106



- Using Help in Hawk 107
 - Indexing Help Files 107
 - Setting Up Hawk to Integrate with MSVC 6.0 Help Files 107
 - Setting Up Hawk to Integrate with MSVC 4.0 (.MVB) Help Files 108
- Printing 108
 - Print Configurations 109
 - Paper Selection Override 109
 - Color Printing 109
 - Print Preview 109
 - Multi-Copy Printing 110
 - Printing Line Numbers 110
 - Wrapping Long Lines in Printed Documents 110
 - Wrapping Long Lines in Printed Documents 110
 - Print Headers and Footers 110
 - Formatting Headers and Footers 110
 - Examples 111
 - Side-by-Side Difference Printing 111
 - Modified Print Dialog 112
 - Functionality 112
- Projects, Project Spaces, and Workspaces 113**
 - Concepts of the Project Menu 114
 - Project Space 114
 - Project Files 114
 - Project 114
 - Workspace 115
 - Component 115
 - Component Options 115
 - Build Order 116
 - Unit 116
 - Unit Maintenance 116
 - Project Preferences 117
 - Creating a Project Space 117
 - The Project Properties List Box and Project Settings 118
 - Project Properties List Box 119
 - Default Settings 119
 - Working Directory 119
 - Setting Project Defaults 120
 - Creating and Modifying Projects 120
 - Adding Files to a Project 122
 - Adding New Projects Using the Members Tab 125
 - Reading External Makefiles and Workspaces 125
 - Steps for Setting up New Makefile/Workspace Parsers 126
 - Characteristics of the File View Tab of the Project Window 127
 - The Directories Tab of the Project Properties Dialog 128
 - Reading Configuration Settings from Other Files 131
 - The Tools Tab 131
 - Tool Categories 132
 - Custom Tools 132



VCS Tools.....	133
Setting up Project Tools.....	134
Command Options on the Tools Tab.....	134
Filename Component Macros.....	135
Symbolic Macros.....	137
The Errors Tab of Project Properties.....	138
Custom Error Parsers.....	140
Traversing the Output.....	141
The Filters Tab.....	142
Project Setup Checklist.....	143
Using Projects and Project Spaces.....	143
Selecting or Changing Projects.....	143
Selecting or Changing Project Spaces.....	144
Loading Files for Editing.....	144
Creating, Selecting and Saving Workspaces.....	144
Creating a New Workspace.....	144
Automatic Saving.....	145
Loading an Existing Workspace.....	145
Searching Project Files.....	146
Selecting files for Check-in or Check-out.....	146
Configuration and State Hierarchy.....	146
The Hawk Debugger 149	
Overview.....	150
Starting the Hawk Debugger.....	150
User-State Applications.....	150
System-State Routines.....	150
The Hawk Debugger Interface.....	151
Hawk Debugger Functionality.....	152
File Functions.....	153
Commands.....	157
View Options.....	159
Browser Tool.....	163
Expression Eval Tool.....	166
Debugging User-State Applications.....	167
NDP Server Functions.....	168
NDP I/O Server.....	169
Connections.....	169
Debugger/NDP Server Connection.....	169
NDP Server/Target Process Connection.....	169
NDP I/O Server/Pseudo Keyboard Connection.....	169
Target Process/Pseudo Keyboard Connection.....	169
NDP I/O Server/Debugger Connection.....	169
Necessary Software.....	170
Required Modules.....	170
Debugging System-State Routines.....	170
NDP Server Functions.....	171
Connections.....	172
Debugger/NDP Server Connection.....	172
NDP Server/Meta Process Connection.....	172

Restrictions.....	172
Communications Layer for System-state Debugging.....	172
Required Modules.....	172
Debugging User-State Applications Using LLROM.....	173
NDP Server Functions.....	174
Connections.....	174
Debugger/NDP Server Connection.....	174
NDP Server/Target Process Connection.....	175
Restrictions.....	175
Necessary Software.....	175
Required Modules.....	175
Installing the Debug Extensions Module for 68K Systems.....	175
Version Control 177	
Using Version Control in Hawk.....	178
Version Control Menu.....	178
Source Code Revision Control—Maintenance Dialog.....	178
Version Control and Hawk Projects.....	179
Associating Version Control Projects with Hawk Projects.....	180
Add Version Control Project Files to a Hawk Project.....	181
Add Hawk Project Files to an SCC Provider Project.....	182
Current Project Tree List.....	182
VCS and the User-Defined Popup Menu.....	183
Modifying the Standard Popup: A Simple Example.....	183
Making Your Own Version Control Popup Menu.....	184
Using Multiple Configuration/Project Files (DOS VCS Utilities Only).....	184
Version Control Integration Configuration.....	185
Version Control for Use with a Command Line Version Control Provider.....	185
Adding a New Command Line Provider to Version Control Setup Dialog.....	186
Customizing Version Control Commands.....	186
Additional Tips.....	189
Hawk SCC Integration with Version Control Systems.....	189
Version Control for Use with a Source Code Provider DLL.....	190
Location of the SCC Provider DLLs.....	191
Search and Replace and Other Navigational Tools 193	
Search and Replace and Regular Expressions.....	194
Search and Replace Dialog.....	194
Search and Replacement Edit Boxes.....	194
Save Settings.....	194
Multiple Sources Search Dialog.....	194
Search and Replacement Edit Boxes.....	195
Current Directory.....	195
Multiple Source Options.....	196
File Pattern.....	196
Search List.....	196
Search Subdirectories.....	197
Edit Modified Files.....	197
Threaded.....	197
Send Listing to Output Window.....	197
Edit Search List.....	198



Search Pattern	198
Patterns List.....	198
Drive and Directory Lists.....	198
Include Directory	199
List Editing Buttons.....	199
Default Button	199
Default Options	199
Search Direction.....	200
Range	200
Prompt	201
Search Options.....	201
Matches.....	201
Start.....	202
Example: Multi-Source Search	202
Incremental Searching.....	203
ISearch Function.....	203
Quick Search	204
Toolbar Search.....	204
Regular Expressions	204
Special Characters.....	205
Escape Sequences.....	206
Matching a Character.....	206
Character Classes.....	207
Escaping Characters in a Class.....	207
Iteration Qualifiers.....	208
Regular Expressions: Positioning at Beginning/End of Line.....	208
Alternation and Grouping.....	209
Reference Groups and Replacement Strings.....	209
Placing the Cursor	210
Searching for Spaces, Tabs and other Blank Characters	211
Searching for Spaces or Tabs.....	211
Searching for New lines	211
Searching for Control Characters (binary/hex data).....	212
Searching for New lines: Issues.....	212
Selective Display.....	213
Selective Display Options	213
Pre-processed View.....	214
Selective Mode.....	214
Browsing, Tags and Symbols.....	215
Using Navigational Tools	215
Browser	215
Tags.....	216
Outline Symbols.....	216
Objects Window.....	216
Browser Support.....	216
Selecting a Database.....	217
Traversing the Tree	217
Label Bitmaps	217
Browser Toolbar	217

Jump to Code	218
String Search	218
Query Button	218
Quick Search	219
Called By/Calls To	219
Search After Go To	219
Filter Toggle Buttons	220
Tags Support	220
Tags Setup	221
Using the Tags Database	221
Outline Symbols: Overview	221
What are Symbols?	222
Symbol Scanning	223
Outline Window	223
Outline Scanning	224
Symbol Parsers	225
Symbols Database	226
Pop-Up Symbols Menu	226
Objects Window	227
Display an Object Hierarchy	227
Using the Objects Window	228
Objects Window and CodeSense	228
Objects Window Popup Menu	228
Symbols vs. Objects vs. Tags	230
Pros and Cons of Tags, Browsers, Symbols, and the Objects Window	230
Bookmarks	230
Global and Local Bookmarks	231
Graphical Bookmark Images	231
Setting and Removing Bookmarks	231
Bookmarks Dialog	231
Bookmarks Window	232
Bookmarks Window: Global Bookmarks	232
Bookmarks Window: Local Bookmarks	233
Bookmarks Window: "Other Documents" Node	233
Auto-Expand/Collapse	233
Button Links	233
How it Works	233
What you See	233
Defining Buttons	234
Checking and Formatting Files 235	
Differencing	236
Interleaved Differencing	236
Difference Analysis Dialog for Interleaved Differencing	236
Interleaved Document	237
Side-by-Side Differencing	238
Difference Analysis Dialog for Side-by-Side Differencing	238
Side-By-Side Difference Window	239
Side-by-Side Difference Printing	242
Directory Differencing	242



Directory Difference Analysis Dialog.....	243
Tree Difference Window	244
Directory Difference Report	246
Using Difference Utilities	247
Merging.....	247
Using the Merge Files Dialog.....	247
Merge Output.....	248
Removing Changes with Merge.....	249
Format Source.....	250
Setting up Your Formatting Criteria.....	250
Using the Format Feature.....	251
Customizing the Interface 253	
Duckbill Toolbars and Windows.....	254
Toolbars.....	254
Auto-hide Toolbars	255
Duckbill.....	255
Enabling and Disabling Toolbars.....	255
Docking and Moving Toolbars and Windows.....	256
Docking a Toolbar or Window Manually.....	256
Undocking a Toolbar or Window.....	256
Customizing Toolbars and Buttons.....	257
Adding New Toolbars.....	257
Adding and Changing Toolbar Buttons.....	257
Binding a Function to a Button.....	258
Combo Box History Lists.....	259
Editing Combo Box History Lists.....	260
Customizing Menus.....	260
Menu Editor	260
Menus	261
Menu Items and Submenus.....	261
Changing the Functionality of a Menu Item.....	262
Adding a Menu Item.....	263
Customizing External Operations within Hawk	264
User-Definable Popup Menus.....	265
Editing or Creating a Popup Menu.....	265
Popup Menu Semantics.....	267
Creating a Menu Item.....	268
Terminating Menu Sections.....	268
Conditionals.....	268
'Include' mechanism.....	269
Dynamic Menu Generation.....	269
Supporting Functions.....	269
Using Keymaps.....	270
CUA Key Commands.....	270
BRIEF Key Commands	271
Mouse Commands.....	271
Mouse Scrolling Speed	271
Inclusive or Exclusive Selection.....	271
Closed Selections	272

Column Marking.....	272
Line Selections.....	272
Word Selections.....	272
Status Line Actions.....	273
Text Drag and Drop.....	273
Mouse Copy and Move.....	274
Creating Windows with a Mouse.....	275
Drag-and-Drop File Loading.....	275
Expand / Collapse Selective Display.....	275
Reassigning Keys and Mouse Actions.....	275
Keymap-Specific Assignments.....	276
Customizing with Keybindings.....	276
Keystroke Recording/Playback.....	276
Recording a Keystroke Macro.....	276
Saving a Macro.....	276
Binding Keystrokes to Functions or Macros.....	277
File Loading and Backup 279	
File Loading, Reloading and Validation.....	280
File Backups and Auto-save.....	282
Making Files Read-only.....	289
Working with Large Files 293	
Swap Blocks.....	294
How to Change the Number of Swap Blocks.....	294
Block Size.....	294
Consider Your Resources.....	295
Backup Files.....	295
Scroll Bars.....	295
Preloading Files.....	296
Turning off ChromaCoding.....	297
Saving Large Files.....	298
Default File Saving Method.....	298
File Rewrite Save Method (for Files over 1MB).....	298
Extending Hawk 301	
Hawk API.....	302
Using the API from the Command Key.....	302
Displaying Return Values With the Command Key.....	302
Examples of Command Key Usage.....	302
Command Key Expression Evaluation.....	303
Macros and Macro Languages.....	304
DLL Extensions.....	305
Where it is Defined.....	306
Perl.....	307
Getting Started with Perl.....	307
Creating and Editing Perl Scripts.....	307
Perl Window.....	308
Pop-up Menu and Options.....	308
Online Help.....	309
Loading and Running Scripts.....	309
Running a Script Directly.....	309



Loading a Perl macro.....	309
Accessing Hawk Functions from Perl Scripts.....	310
Importing Names into the Perl Namespace.....	311
Unloading a Perl macro.....	311
Using the Perl Debug Mode.....	312
Accessing Perl Functions.....	312
Avoiding Ambiguity.....	312
Special API Functions for Perl.....	313
Files Used by Perl for Hawk.....	314
Other Perl Resources.....	315
AppBasic.....	315
Getting Started.....	316
Two Editors.....	316
Special Keybindings.....	316
Two Toolbars.....	317
Pop-up Menu.....	318
Online Help.....	318
UserDialog Editor.....	318
Object Browser.....	318
Creating a Macro.....	318
Creating a Handler.....	319
Object and Proc Dropdown Lists.....	320
Private Sub Main.....	320
Tips on Creating Macros.....	321
Creating a Modal User Dialog.....	321
Running the Macro.....	324
Creating an EventHandler in AppBasic.....	324
Sample EHTEST.CWB.....	325
Debugging Your AppBasic Macro.....	326
Break Points.....	326
Evaluate Expression and Add Watch.....	327
Object Browser.....	327
Load Macros Dialog.....	328
AppBasic Samples Supplied.....	329
AppBasic-related API Commands.....	331
AppBasic Window Configuration.....	332
Example Configuration File Settings.....	333
Exported Functions in CWBASIC.DLL.....	333
API (C-like) Macros.....	335
API Macros Defined.....	335
Getting Started with API Macros.....	335
Creating a Macro.....	336
Editing a Macro.....	336
Special Editing Keystrokes.....	336
Editing API Macros in a Hawk Window.....	336
Setup Editing in a Hawk Window.....	337
Language Definition.....	337
Comments.....	337
Identifier Naming Rules.....	338

Variables.....	338
Literal Values.....	339
Expressions.....	341
Other Operators.....	343
Statements and Statement Blocks	346
Program Flow of Control Structures.....	346
Run-time Error Handling.....	348
Hawk Event Handling.....	349
Differences between API Macros and C.....	349
String Functions	350
Making DLL Add-Ons	351
Core Services.....	352
CWSTART DLL.....	352
CWDIALOG DLL	356
CWHELP DLL.....	356
Keyboard Command Sets.....	357
Supplemental Language Support.....	357
Auxiliary Services.....	358
Sample DLL.....	359
Dissecting a Hawk DLL	360
The _init Function.....	360
Exporting Functions	360
Making Changes and Additions.....	360
Changing Existing Functions	360
Adding Your Own Functions.....	361
Creating New Keymap Command Sets	361
Keymap _init Function.....	361
Keymap Function	361
Flag Initialization.....	362
Basic Assignments.....	362
Keymap Specific Assignments.....	362
Menu Accelerators	362
Recompiling a DLL	363
Using and Modifying the Makefiles.....	363
Adding Files.....	364
Compile and Link Options	364
Link Libraries	364
Microsoft Link.....	364
Borland Link.....	365
Using Your Own DLL.....	365
Installing Your DLL.....	366
UNIX 367	
File Type and End-of-Line (EOL) Characters	368
Save File as New Type.....	368
Make UNIX EOL Characters the Default.....	368
Enable Auto-sense File Type Option.....	369
Specify UNIX EOL Characters on a File Type or Per File Basis.....	369
Change EOL Characters in the Source File.....	370
Compiling UNIX Programs from Hawk	371



Filename Case and File Securities in UNIX and Windows	371
Configuration Files and Command Line Parameters 373	
Location of the Configuration File	374
Introduction to Configuration and State	374
Configuration File	375
State File	375
Other Files Containing Configuration Data.....	375
Example File.....	376
Example Interpretation	377
Processing At Startup.....	378
Order of Processing	378
User-Defined Sections.....	379
Relating Check Boxes to Functions.....	381
State File	383
Location of the State File.....	383
Contents of the State File.....	384
Filenames.....	385
Parameters.....	386
Command Files.....	390
Using Hawk™ Profiler 391	
Setting up the Target System	392
The Profiler	392
Running the Profiler.....	392
The Profiler Main Window	392
Title Bar.....	393
Menu Bar.....	393
System Performance Tab	394
Module Data Tab	394
Start/Stop Button	394
Connect/Disconnect Button	395
File Menu.....	395
Options Menu.....	395
Actions Menu.....	396
Using the Profiler	396
Viewing Functions	396
Saving the Profile Data.....	396
Changing the Order in which Modules/Functions Appear.....	397
Profiling Selected Modules.....	397
Changing the Update Interval	397
Stopping Profiling.....	397
Clearing the Profiling Data	397
Exiting the Target System Profiler	397
Using HawkEye™ 399	
Starting HawkEye	399
TagsWNN Utility 401	
Usage.....	401
TagsWnn Command Line Options	401
“Null Modem” Cable for Windows	

SLIP 407	
Wiring Diagram	407
Command I/O 409	
Commands	410
asm,	410
a[ssign]	410
at[tach]	411
b[reak]	411
con[text]	411
dil[ist]	411
di[sasm]	411
download	412
d[ump]	412
fi[nd]	412
fo[rk]	412
f[rame]	412
g[o]	413
gostop or gs	413
i[nfo]	413
k[ill]	413
l[ist]	414
lo[cals]	414
l[og]	414
mc[opy]	414
mf[ill]	414
mfillong	415
mfillword	415
m[onitor]	415
ms[earch]	415
msearchlong	416
msearchword	416
n[ext]	416
ow	417
owa	417
owd	417
owk	418
p[rint]	418
pr[ocs]	418
q[uit]	418
re[ad]	418
rel[ease]	418
reset	419
r[eturn]	419
se[te]nv	419
s[tep]	419
sy[mbol]	420
t[race]	420
unse[te]nv	420
wait	420



wfi[nd].....	420
x{string}.....	421



1

Introduction

This chapter provides an overview of the Hawk™ integrated development environment, and includes the following sections:

[Hawk Overview](#)

Hawk Overview

Hawk is a comprehensive software development and management environment that increases the efficiency of OS-9® development. The integrated tool enables you to create, debug, analyze, and manage OS-9 real-time software development projects.

Hawk features the following integrated tools:

The Hawk Project Manager

The Hawk project manager enables programmers to configure software at any level. All build settings are inherited by default (group source files, makefiles, libraries, etc.). Some of the project manager's main features include the following:

- software project organizer
- identifies dependencies between software components
- records detailed build information
- controls the build process

The Hawk Editor

The Hawk Editor is a modified version of Starbase Corporation's Codewright editor. The editor provides an organized, fully programmable interface for creating and maintaining OS-9 projects.

Table 1-1 provides a list of key editing features.

Table 1-1. Hawk Editing Features

Category/Chapter	Feature
Introduction	Hawk is a flexible, multiple-window, multiple-file interface. Edit as many files in as many windows as your resources allow.
Running Hawk for the First Time	New Each tab of the Output Window may be hidden or made visible on the Customize -> Toolbars -> Show Tabs dialog.
	Configuration Wizards assist you in setting up the Help Index file and more. Also included is an Answer Wizard.
	Support is available for Microsoft's HTML Help Viewer.
	There is a customization shortcut on the status bar; this allows you to quickly access many of the items you want to configure.

Table 1-1. Hawk Editing Features (Continued)

		<p>The Document/Window Manager has a Window tab that consolidates the settings that affect individual documents and windows.</p>
		<p>The Wrap option on the Display tab of the Document/Window Manager allows you to wrap text on the display only, adjusting the wrap column automatically when you resize the window. Check Column and enter a column number to wrap at a specified column. Check the Words box to avoid splitting words at the wrap column.</p>
Command Key, Libraries and Environment		<p>Unicode Support. You can turn on auto-detection and bi-directional conversion of code that follows the Unicode standard by marking Auto-detect Unicode files on the Customize -> Environment -> General dialog.</p> <p>The Windows API function <code>wideCharToMultiByte()</code> is used to convert Unicode to ANSI Code Page characters when a file is loaded and then <code>MultiByteToWideChar()</code> is used to convert back to Unicode when the file is written out.</p> <p>Unicode File Conversion is also available on request. From the File -> Save As dialog, you can save "Normal Text" files as Unicode, or Unicode files as "Normal Text".</p>
View Setups and Language Support		<p>Tile specified windows horizontally, vertically, or in one row from the Document/Window Manager.</p>
View Setups and Language Support	New	<p>A new lexer and CodeFolio Snippets are available for the Vignette StoryServer Tool Command Language:</p> <p>On the Customize ChromaCoding Lexers dialog, select Vignette TCL from the Lexer list.</p> <p>On the CodeFolio tab of the Project Window, expand the vtcl Snippets directory.</p> <p>You may also download additional Code Snippets. Refer to applicable sections of the Web Library document How To: Setup "Vignette TCL" ChromaCoding Lexer and CodeSnippets for Vignette StoryServer.</p>
		<p>Interactively create ChromaCoding lexers to color the syntax for your programming language.</p>
		<p>Use View Setups for storing sets of colors, fonts, and window attributes under individual names.</p>

Table 1-1. Hawk Editing Features (Continued)

	Background and foreground palette colors can be changed (up to 16 million colors, if supported by your display).
	Associate a View Setup with a particular file type on the updated Language dialog.
	Use color by ChromaCode to signal changed lines or highlight language syntax.

Editing Features	New	The Additional Clipboard Formats and Options on the Customize -> Environment -> Clipboard dialog allow you to save clipboard content in RTF and/or HTML format (in addition to text format), replace tabs with spaces, and preserve background color settings in the clipboard content from the current View Setup.
		Multiple Clipboards/Scrap Buffers allow you to copy more than one item for later use without overwriting previous items. An auto-increment clipboard option increments the clipboard/scrap buffer to be used prior to copying or cutting the new content.
		Clipboard and Scrapboard viewer are now available in the ClipView tab on the Output window for viewing and selectively pasting items in the current buffer.
		Enhanced command line editing allows you to select text on the command line for cut and paste. Keyboard shortcuts are available for the command line prompt.
		CodeSense parses programming libraries and project source files to provide word completion for symbols, function parameter help, and symbol definition help.
	New	CodeSense for C#
	New	C++ Namespaces are supported by CodeSense; a Namespaces filter has been added to the Filter/Legend dialog and to the popup menu available from the Objects Window.
	New	CodeSense is able to parse source files inside .zip or .jar files (a .jar file is like a .zip file for Java).

Table 1-1. Hawk Editing Features (Continued)

New	<p>Brace Matching functions have an expanded definition of braces. In addition to recognizing and locating curly braces, square brackets, and parentheses, language-specific definitions allow Hawk to match various brace/tag pairs.</p> <p>For Pascal, the keywords <code>Begin</code> and <code>End</code> are treated like braces; Hawk looks for <code>Begin</code> and finds the matching <code>End</code>. For HTML/XML, tags are also included; Hawk considers the current tag an open brace and finds the matching tag.</p>
New	<p>The Customize ->CodeSense Global Configuration dialog now allows you to Enable Extended Popup Hotkeys for use with CodeSense and the Objects Window; adjust the Maximum Number of Hits that a CodeSense-related lookup will return; and view the Properties of the CodeSense library database currently highlighted</p>
New	<p>A new Print button and modified Print dialog let you print from the Side-by-Side Difference Window in a manner similar to two-up mode, except the left side will contain the Reference document/file and the right side will contain the Target document/file.</p>
	<p>CodeFolio Snippets is a new <code>CodeFolio</code> tab on the Project Window. It provides a directory of code samples that are specific to programming languages and can be inserted into the current document. Users may add snippets to the directory.</p>
	<p>Zoom functionality is now available on the <code>Print Preview</code> dialog.</p>
	<p>Hex mode for editing binary files, including inserting and deleting bytes, is available.</p>
	<p>Unlimited Undo and Redo of commands (from the <code>Edit</code> menu) are available.</p>
	<p>Powerful templates for language constructs, personalized function headers, and more are available. Any Hawk API function that can be executed interactively can be executed within a template.</p>
	<p>API Assistant helps you accurately complete function calls by displaying the types of and options for each parameter defined for the function. After you check the desired boxes, it then constructs the proper call for insertion into your text.</p>

Table 1-1. Hawk Editing Features (Continued)

Projects, Project Spaces and Workspaces		Multiple projects may be displayed at a time within a project space.
Search and Replace and Other Navigational Tools	New	.You can have up to five Search tabs visible on the Output Window (Search 1 is the default).
		Save and restore cursor position when closing and re-opening documents
	New	The Objects tab on the Project Window has new buttons that toggle whether to Ignore Case or Include CodeSense Library Databases in a lookup. New buttons also access the Filter/Legend dialog and context menu.
		Complete support of regular expressions in searching and replacing.
		Multi-document or file-based search and replace is available, including searches across multiple lines and replacement groups.
		Button Links let you reference and view external documents, such as diagrams and word processor files, or organize notes into a list. These links appear as graphical buttons within the text file.
		Several types of browsing, including Tags and Outline Symbols are available.
		Unlimited local and global bookmarks are available.
Checking and Reformatting Files	New	The Directory Difference Analysis dialog lets you compare the files in two directories, with or without subdirectories. A tree directory displays on the Difference tab of the Output Window; double-click on any file that exists in both directories to perform Side-by-Side Differencing on that file.
	New	After performing Directory Differencing, right-click on any file in the tree directory and select Copy File to copy the file from one directory to another.

Table 1-1. Hawk Editing Features (Continued)

	New	Once Directory Differencing has been performed, see analysis settings, file and directory counts and a list of files in the directory tree on the Directory Difference Report. Customize the report using the API macro DiffDirReport.
		Expanded spell check dialog and additional dictionaries for languages other than English are available.
		The spell checker will check the spelling of source code comments and strings only, if desired. You can spell check the whole buffer, or just the selection.
		Side-by-side Differencing, Difference Editing, and Merging of files are supported.
Customizing the Interface	New	Auto-hide toolbars are available, as are auto-hide menus in Full Screen mode.
		Interactively definable Toolbars are available for quick mouse access to common commands.
		Selective Display mode is available for selecting which lines to make visible or invisible. It also allows selected lines to be viewed with grep-like commands, or by preprocessing <code>#ifdefs</code> .
		Use Windows' features, including resizable Common Dialogs for familiar operation, Drag and Drop for convenient file loading, e-mail integration, and Tabbed Dialogs (property sheets) to simplify dealing with settings and options.
File Loading and Backup		Auto-save limit allows you to limit the size of files that are auto-saved.
		Additional auto-save features on the <code>Environment -> Backup</code> dialog for controlling the directory and file extension to be used for auto-saved files.
		Optional auto-save is available at scheduled intervals or during idle periods.
Working with Large Files		Line length and file size virtually unlimited.
Extending Hawk		Enhanced API Macro Language is supported.
		CUA, BRIEF, Epsilon, or vi style key commands are available.

Table 1-1. Hawk Editing Features (Continued)

		A completely remappable keyboard is available; you can create your own command set.
		Hawk is configurable and extensible through Macros and DLLs.
End of Line characters	New	<p>You can specify the type of end-of-line characters that are inserted into a new file (DOS, Unix, or Mac), and even choose to convert existing characters to the new type. These EOL options can be found in the following locations:</p> <ul style="list-style-type: none"> • On the Options tab of the Language dialog, to specify an EOL style on a file-type (e.g. .C, .TXT, HTML, etc.) basis. • On the Options tab of the Window Manager dialog, to specify an EOL style on a per-document basis.

The Hawk Debugger

Hawk features a fully integrated debugging tool for efficient debugging of C and C++ code. Some of its main features include the following:

- source- and assembly-level breakpoints
- display/change registers
- view locals
- watchpoints
- directly view/change memory
- stack back tracing
- easy-to-use interface
- system- or process-level debugging
- debug forked child processes and threads

In addition, the Hawk tool set features and full-function, stand-alone debugger, which can be started and run outside of Hawk.

The Hawk Compiler

Customize the Ultra C/C++ compiler settings using a point-and-click, pull-down menu format. Microware Ultra C/C++ is a highly optimizing compiler that uses state-of-the-art optimization techniques to extract maximum performance from modern 16- and 32-bit RISC and CISC microprocessors. Most compilers only let the user optimize the application on a file-by-file basis. Ultra C/C++ can see and optimize the application, together with libraries, as a whole.

2

Running Hawk for the First Time

This section describes what you will see when you run Hawk for the first time. The following sections are included:

[A First Look](#)

[The Menu Bar](#)

[Output and Project Windows](#)



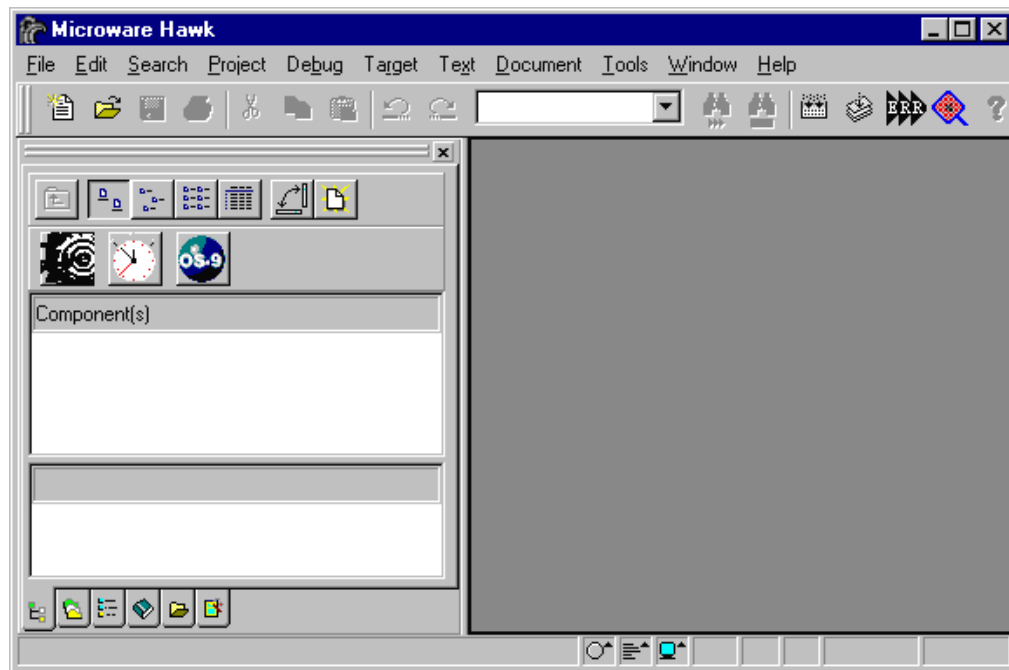
If you would like to run Hawk from one server, rather than multiple workstations, you can do so from your Microware OS-9 product CD. Simply open the main installation window, select **Install Microware OS-9 for <your product>**. Once the InstallShield brings up the Installation Options dialog, select **Client Install**, then follow the remaining procedures. This will install only those components necessary to access a server installation of OS-9.

Under the Client Install option, Hawk uses the value assigned to the HAWKCLIENT environment variable as the path to local files and the MWOS directory as the path to the server installation. The files that reside in the local directory are user configuration files. Any files not found in this directory are opened on the server path. Additionally, "created files" (such as `mwhawk.pst` and `hawkdata.xml`) are placed in the local directory, and the installer-created PROJECTS directory is placed inside it. If HAWKCLIENT does not exist, Hawk uses the path specified in the MWOS environment variable to open all files.

A First Look

The initial Hawk screen (shown in [Figure 2-1](#)) displays a menu bar at the top, with the Standard toolbar docked directly beneath. The window on the left is the Project window; the window on the bottom left is the Output window. These items will be described briefly throughout the remainder of this chapter.

Figure 2-1. Initial Hawk Screen



The Menu Bar

The menu bar consists of the nine items described in the following sections.

File Menu

The first menu is the File menu. The File menu is intended for standard file operations, such as New, Open, Save, Print. Additional items are described in other chapters of this manual or explained below:

- The Difference and Merge utilities are described in detail in [Chapter 11 Checking and Formatting Files](#).
- The Find menu item is used for finding files on a storage medium. Standard DOS wildcard characters can be used when using this option to search for files.
- The Send Mail selection sends files via MS mail or cc:Mail.
- The Reload item reloads the current file from disk.
- The Change Directory item accesses a dialog that allows the current directory to be changed.

- The `Filters` dialog defines file specifications or wildcard patterns, such as `*.c`, to filter files of interest from otherwise long lists of files. It is a good idea to set the filters during the initial Hawk session so they will be ready to use in the future. [Table 2-1](#) shows some useful filters.

Table 2-1. Filters

Description	Pattern
text files	*.TXT;*.RTF
Initialization files	*.INI;*.CFG
Make files	*.MAK;*.

To store filter settings with the current project, check the `Save Filename Filters in project file` box. If the box is checked but no project is open, the settings are stored as default settings to use with future projects.

- The `Change Directory` dialog allows you to set a new working directory for Hawk. If you have elected not to check the `Set as CWD` check box in the `File Open` dialog, you may, on occasion, want to change directories using this mechanism.
- The `Print` and `Print Setup` dialogs are described in detail in [Chapter 6 Editing Features](#).

Edit Menu

The first two sections of the `Edit` menu contain standard editing items, such as `Undo`, `Redo`, `Cut`, `Copy`, and `Paste`. The following lists items with which you may not be familiar:

- The `Scrap Buffer` item selects the `Scrap Buffer` for cut/paste operations.
- The `Append` item adds selected text to the end of the selected clipboard/scrap buffer.
- The `Erase` item clears the selected clipboard/scrap buffer.

Hawk's Scrap and Clipboard features are described in detail in [Chapter 6 Editing Features](#).

The next half of the `Edit` menu, shown below, contains options for inserting items into the current buffer:

- The `Insert File` selection brings up a dialog for selecting a file to insert into the current document.
- The `Insert Literal` item brings up a dialog for entering characters into the document that would otherwise trigger a command. The literal may be chosen using ASCII, Hex, or Decimal values.
- The `Insert Link` dialog inserts a button link at the cursor position in the current document. Button links are special action buttons that can be embedded in your text files. You can use them to view bitmapped images, bring up a related document or spreadsheet, run a macro, or make notes.
- The `View Links` item is used for viewing the contents of the `Button Links` database.

Button Links are described in detail in [Chapter 10 Search and Replace and Other Navigational Tools](#).

- The Record, Playback, Keystroke Macros, and Run Key Macro are for making, testing, editing and using Keystroke Macros, respectively. Hawk's Keystroke Macros are described in detail in [Chapter 12 Customizing the Interface](#).

Search Menu

The Search menu contains all the items necessary for performing search and replace operations on single or multiple files. All of Hawk's search features are described in detail in [Chapter 10 Search and Replace and Other Navigational Tools](#).

The Options item on the Search menu brings up the Default Search and Replace Settings dialog; in this dialog you can set up a specific way to operate searches. It is important to set the search options before editing so that some of the following issues might be addressed:

- Should the search be case sensitive?
- Should there be a prompt on replacement by default?
- Should regular expressions be found?
- Should matching text be selected (highlighted)? If so, should the text be selected momentarily or continuously? If it is selected continuously, should Restrict to Selection be turned off so Search Again can work?
- Should the word upon which the cursor sits be typed or located in the Search dialog by default?

Project Menu

The Project menu is used for creating, opening, closing and manipulating projects, project spaces, and workspaces. When working with the Project menu, keep the following in mind:

- Projects organize and store individual sets of files and configuration settings.
- Project spaces organize sets of projects.
- Workspaces are "mini-projects" within projects.

Options on the Project menu include the following:

- The Project Space option displays a submenu containing New, Open, and Close, for adding new or existing projects and for removing projects from the project space. Project -> Set Current is for opening a project within a project space.
- The Compile, Compile (Debug), Build, Rebuild, Rebuild (Debug), Debug, and Execute options must be configured in the Project -> Properties dialog in order to be used.

More information about Hawk projects is provided in [Chapter 3 Building Your First Project](#) and [Chapter 7 Projects, Project Spaces, and Workspaces](#).

Debug Menu

The Debug menu contains items necessary for debugging modules. Some of the more pertinent menu items include the following:

- **Connect** : This option allows you to select the target name and location of the module you want to debug.
- **Options**: This menu item allows you to set session and animation options, location of code to be debugged, and target environment debugging options.
- **View**: This option toggles the visibility of various windows, such as breakpoint, memory, and stack.
- **Process** : This menu item offers options pertaining to the current process, such as reforking and attaching modules.

Target Menu

The Target menu contains items useful for working with targets.

- **Load**: This option loads modules into the target's memory.
- **Unlink**: This option reduces the link count of the module.
- **Download bootfile**: This option downloads a bootfile to the target machine.
- **Ident Module**: This option performs an ident of a module on the target machine. The ident utility displays such information as module size and revision number.
- **Emulator** : Start the Emulator Program
- **HawkEye** : Open the HawkEye Interface (optional Add-on product)
- **Profiler** : Profiles processes/threads or modules on the target system.
- **Wizard** : Brings up the OS-9 dialog
- **TechCheck** : Brings up the Microware Tech-check tool. Used to help collect system information for technical support calls.

Text Menu

The Text menu has various items for formatting and editing text. Some of these items, such as **Selective Display**, **Comment**, and **Format Source**, are discussed in various chapters. The other items are briefly described in this section.

- The **Word Wrap** option enables and disables automatic line wrapping. The lines wrap at the right-margin mark, which is set on the **General** tab of the **Customize -> View Setups** dialog. On that dialog, if **Wrap Display Mode** is selected, but **Wrap Column** is not, the right-margin is automatically adjusted when the window is resized.
- The **Upper** and **Lower** menu items capitalize or un-capitalize selected text.
- The **Slide In** and **Slide Out** menu items slide the current line or the current selection right (in) or left (out), according to the length of a tab stop. The

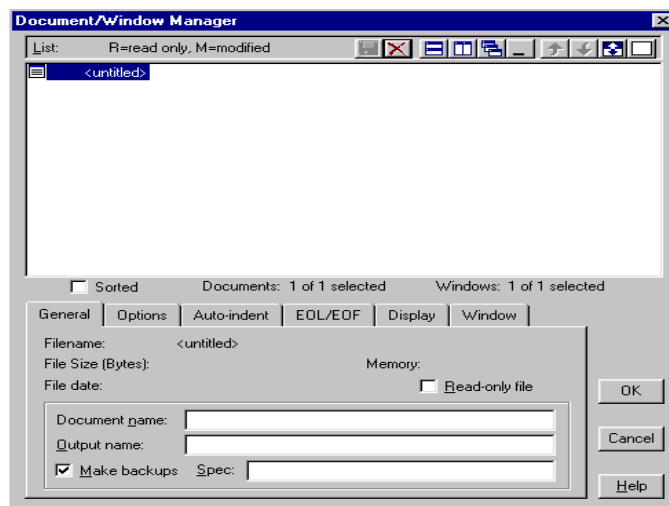
Prompted Slide In/Out items prompt the user for some text that slides text in or out.

- The Left Justify Right Justify and Center items position text within a column or line selection.
- The Enumerate menu item inserts line numbers in ascending or descending order for selected text.
- The Format Columns option aligns columns within a selected block.

Document Menu

The Document menu contains items that access dialogs and perform operations commonly used on documents. The Document/Window Manager dialog (shown in Figure 2-2) lists documents and windows currently open in Hawk. It can be used to change the attributes of open windows and documents on an individual basis.

Figure 2-2. Document/Window Manager Dialog



Some of the operations that can be done in Document/Window Manager include the following:


- specify indent options
- tile/cascade individually selected documents or windows
- set View Setups (such as fonts or colors) for individual documents/windows
- change display modes (such as Selective or Hex displays)
- change window attributes (such as scrollbars or line numbers)
- set soft line wrap (located on the Display tab)

This option wraps lines without inserting a line feed and adjusts the wrap column as the window is resized. Check Column and enter a number to wrap at a specified column. Check the Words option to avoid splitting words at the wrap column.

The Customize Menu

The `Customize` menu contains items that access several comprehensive dialogs. These dialogs are used to configure and customize Hawk. Custom tools are set up in the `Project -> Settings` dialog on the `Tools` tab. They are described more comprehensively in [Chapter 7 Projects, Project Spaces, and Workspaces](#).



Many of the items available on the `Customize` menu may also be accessed from the Customization shortcut icon  on the status bar.

The following items are included in the `Customize` menu:

- The `Environment` menu item accesses the `Environment` dialog, which has a number of tabs for manipulating the Hawk environment. For example, setting backup files and backup specifications is done on the `Backup` tab of the `Environment` dialog; modifying menus is done via the `Menu` tab.



Portions of the `Environment` dialog are discussed in [Chapter 4 Command Key, Libraries, and General Environment](#). However, since the tabs in the `Environment` dialog relate to a wide variety of features in Hawk, the items in the dialog will be covered more extensively throughout this manual.

The following two items are vital tabs under the `Environment` dialog:

- **General**

This tab is used for setting various items in the Hawk environment. Concerns addressed by the settings on this tab include the following:

- Should system prompts, such as `API Command` dialog/prompt and pop-up in a dialog be displayed on the status line? Often, the preference is the status line prompt, the setting for which is `Use Command Line Prompt`.
- Should windows and documents operate as a single unit, as they do in most Windows applications, or independently? The toggle for this functionality is `One Document Per Window`.
- Should a list of recently loaded files appear at the bottom of the `File` menu? This option can be used for reloading files; it is turned on by marking `Show File List on File Menu`.
- What are file loading and file validation concerns?
- Should the `Vertical/Horizontal Tile` options on the `Window` menu tile vertically, horizontally, or in the traditional Windows manner.
- **State**

This tab allows you to save state information between Hawk sessions. When using this tab, address the following questions:

- Do you want Hawk to remember and load the last file you worked on, all of the files you worked on, or none of the files you worked on?
- Do you want to save bookmarks between sessions?

- The `Read Configuration Data` menu item on the `Customize` menu accesses a dialog that allows certain configuration settings to be read from any file into the current configuration file. Configuration files contain settings that tell Hawk how it should look and behave.
- The `Libraries` option is used for interactively loading Hawk DLLs (Add-Ons) to extend the functionality of the Hawk program. A number of commonly used Add-Ons are contained in the list of Hawk Libraries in the `Libraries` dialog. The Hawk Libraries can be conveniently loaded by placing check marks next to those being loaded.

More Add-Ons are available on Starbase's web site, at the following address:

<http://www.premia.com>

- The `Toolbars` and `Keyboard` menu items bring up the `Toolbar Customization` and `Assign Keys` dialogs, respectively. Descriptions for how these dialogs are used can be found in [Chapter 12 Customizing the Interface](#).
- The `Language` option accesses a dialog that is used for controlling language-specific features, such as ChromaCoding (syntax coloring), and template expansion.
- The `ChromaCoding Lexers` option accesses the `ChromaCoding Lexer Settings` dialog. In this dialog you can create or modify lexers to color the various elements of your programming language. The vocabulary includes such items as identifiers, braces, preprocessors, keywords, operators, strings and comments.
- The `View Setups` option accesses a dialog used for viewing and changing document and window preferences, such as colors, fonts, line numbers, and scroll bars.



[Language](#), [ChromaCoding Lexers](#), and [View Setups](#) are described in [Chapter 5 View Setups and Language Support](#)

- The `CodeSense Libraries` option accesses the `CodeSense Libraries` screen. CodeSense completes function names as you type them; they also automatically suggest appropriate parameters to inserted for the function. It is available for C, C++, C#, and Java files.

CodeSense works by parsing the programming libraries that you add on the `CodeSense Libraries` screen, as well as project source files. In order to use this feature, CodeSense must be configured on the `Customize -> Language -> CodeSense` dialog. For more information on CodeSense, refer to [Chapter 6 Editing Features](#).

Tools Menu

The `Tools` menu has a variety of tools for manipulating and extending Hawk. Items on this menu include the following:

- The `API Command` option is described in [Chapter 4 Command Key, Libraries, and General Environment](#)

- The `API Macros` and `Run API Macro` menu items are used for making and running Hawk API Macros.
- The `AppBasic Macros` and `Perl Macros` menu items both have submenus containing items for loading and working with `AppBasic` and `Perl Macros`. Macros are described in detail in [Chapter 15 Extending Hawk](#).
- The `Version Control` item on the `Tools` menu accesses a submenu containing items that either run version control operations or access dialogs used for running or setting up version control.

When looking at this submenu, it is important to remember that Hawk does not come with its own version control utility. Instead, it integrates with existing version control systems using one of two methods: command line integration or SCC API integration. More information about using version control from within Hawk can be found in the chapter on [Chapter 9 Version Control](#).

- The `Filter` option brings up a dialog that allows you perform an external operation on the current document or a selected portion of it.
- The `Shell` option runs a command shell.
- The `Shell Command` option brings up a dialog that invokes a Windows/DOS application.
- The `API Database Editor` command runs the API Assistant Editor.
- The `Paint` command runs Microsoft Paint.

Window Menu

Similar to the `Document` menu, the `Window` menu is used for manipulating and creating windows opened in Hawk. It is important to understand how Hawk differentiates these windows. Depending on the configurations made in Hawk, open files may not always display in their own windows. The difference between windows and documents is described below.

Windows and Documents

Some confusion may arise when attempting to discern how Hawk differentiates windows and documents. Depending on the keymap being used, some documents opened in Hawk may display in their own windows.

Most of the keymap emulations that Hawk provides, such as `BRIEF`, `Epsilon`, and `vi`, use one window for all documents by default. This characteristic defers to the DOS days of editors, from which the referenced keymap emulations were derived. The `CUA` keymap, however, uses one window for every document. Therefore, in `CUA`, each window contains its own document.

The "one document per window..." characteristic can be toggled for any keymap using `One Document per Window` in `Customize -> Environment -> General` under the `General` heading. Keymaps are described in [Chapter 12 Customizing the Interface](#).

Window Menu

- The `New Window` option creates a new window. If a document is already open, the new window will contain another instance of that document.
- The `Full Screen` option puts documents into full-screen mode, eliminating the clutter caused by menus and toolbars and maximizing editing space.
- The `Tile` and `Cascade` options are alternative ways to arrange Hawk windows.
- The `Arrange Icons` option arranges minimized windows at the bottom of the Hawk screen. The `Close All` option closes all windows that are open.
- The `Project` and `Output` options toggle the `Project` and `Output` windows off and on. A brief description of these windows is provided in the next topic, [Output and Project Windows](#).
- The `Manager` option accesses the `Document/Window Manager` dialog. This is the same dialog that is accessed when clicking the `Manager` option on the `Document` menu. For more information about the `Document/Window Manager`.

Help Menu

Hawk's `Help` menu is for accessing and configuring Hawk's online help and API Assistant. More information on `Help` and the API Assistant can be found in [Chapter 6 Editing Features](#).

Output and Project Windows

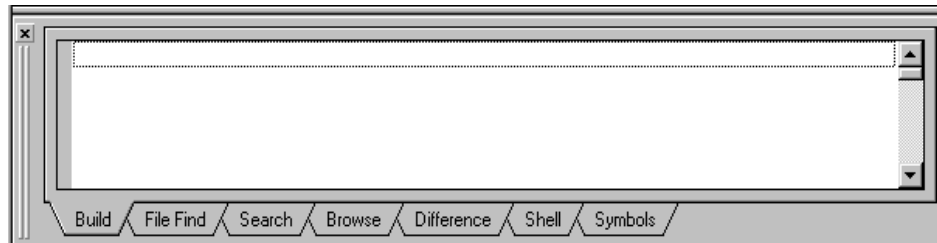
The default Hawk screen has two system windows, including one on the left and one on the bottom. The window docked on the bottom of the Hawk screen is called the `Output window`. The window on the left is called the `Project window`. These two windows can be "undocked" by dragging them away from any edge of the Hawk screen.

- More information on customizing the `Project` and `Output Windows` can be found in [Chapter 12 Customizing the Interface](#).
- The `Project window` is described in detail in [Chapter 7 Projects, Project Spaces, and Workspaces](#).
- The `Output window` contains a number of tabs that relate to various features in Hawk. It is described in detail in various chapters of this manual.

The Output Window

The `Output window` (shown in [Figure 2-3](#)) initially displays seven tabs and is located on the bottom edge of the Hawk screen. You can add three more tabs by loading the Add-Ons for the `AppBasic` and `Perl` macro languages and the Add-On for the `Clipboard/Scrap Viewer`. Add-Ons are loaded using the `Customize - > Libraries` dialog.

Figure 2-3. The Output Window



Uses for the tabs of the Output Window may vary, however, they are generally used for displaying and manipulating the results of operations in Hawk.

Below are descriptions of each tab of the Output Window:

- The **Build** tab displays the results of builds or compiles.
- The **File Find** tab displays the results of file searches produced by **File -> Find**.
- The **Search 1** tab displays the results of string searches produced from search operations. There are five search tabs available for search operations. To make additional ones visible select them in the **Customize -> Toolbars -> Show Tabs** dialog.
- The **Browse** tab displays tags or browser database files.
- The **Difference** tab displays the differences between two files, side-by-side.
- The **Shell** tab acts as a virtual DOS shell.
- The **Symbols** tab allows you to navigate through files. This is similar to the **Browse** tab.

Auto-Increment Search Tabs

To use the visible search tabs sequentially for the output of successive search operations, select the **Auto-Increment Output Tab** check box. This check box is located on the **Search Output Options** dialog box. The **Search Output Options** dialog box is displayed from the **Output** button in the **Search and Replace Multiple Sources** and the **File Grep** dialog boxes.

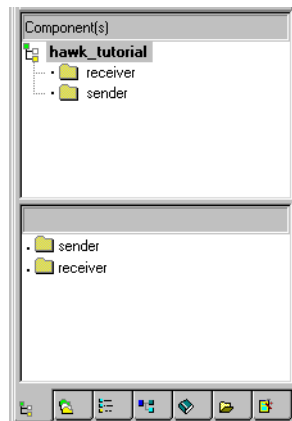
Append to Listing

If the **Append to Listing** check box is checked with the **Auto-Increment Output Tab** check box, the results of the next search will be appended to any results already in the search tab.

The Project Window

The **Project Window**, shown below, is initially displayed on the left edge of the Hawk screen.

Figure 2-4. Project Window

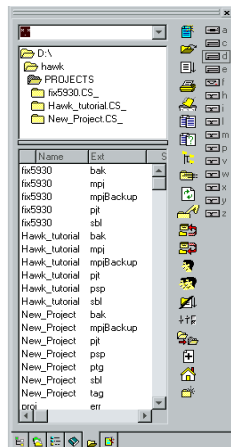


Like the `Output Window`, the `Project Window` contains several tabs. They are for viewing information about projects and other files. The tabs include the following from left to right:

- The `Project` tab displays projects in project spaces, as well as project components.
- The `File View` tab lists the files for the current project in hierarchical form and allows you operate on individual files or groups of files. More information about the `File View` tab can be found in [Chapter 7 Projects, Project Spaces, and Workspaces](#).
- The `Outline` tab displays a hierarchical view of `Symbols` in project files and any other files that are currently loaded. More information about `Symbols` and the `Outline` tab can be found in [Chapter 10 Search and Replace and Other Navigational Tools](#).
- The `Objects` tab is used to view and browse code. It displays a hierarchical view of C/C++, C# and Java objects/symbols that are associated with the name that is typed in the identifier box at the top of the window. C++ Namespaces are supported.
- The `Bookmarks` tab gives a view of local and global bookmarks defined in documents. More information about `Bookmarks` and the `Bookmarks` tab can be found in [Chapter 10 Search and Replace and Other Navigational Tools](#).
- The `Open` tab acts like a persistent `File Open Dialog` and file manager in one. It presents a list of icons representing valid drives, a directory tree, an area in which to specify a file filter (this limits the files displayed), and a box in which matching files are listed. Double-click on any of the files listed to load it for viewing or editing. More information on the `Open` tab is provided in the next section, [The Standard Toolbar](#).

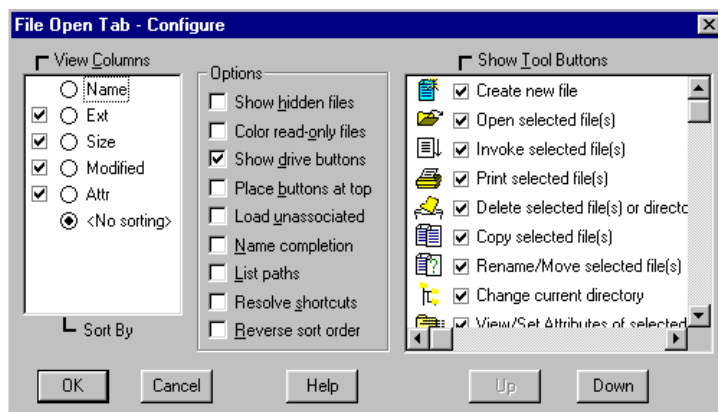
The `Open` tab (shown in [Figure 2-5](#)) can perform a total of twenty operations, as represented by a series of icons on the frame of the `Open` tab. Tool tips (small popup messages) describe each of the icons.



Figure 2-5. Project Window: Open Tab



Use File Open -> Configure dialog to remove, add, or reconfigure the icons on the Open tab. To access this dialog, click on the **Configuration Dialog** button. The Configure dialog displays:

Figure 2-6. File Open Tab -> Configure Dialog



Two of the icons you can set for the Open tab are for user-defined commands:  (User function #1) and  (User function #2). User-defined commands can be set to perform any operation on files selected within the window. The user commands must be Hawk API function calls, rather than DOS, or other similar commands. You may, however, execute DOS commands by specifying the API function `ExecUserCmdnd`. Help with `ExecUserCmdnd` or any other Hawk API can be found in Hawk's online Help.

Example

```
ExecUserCmdnd "Dir"
```

The File Open Tab -> Configure dialog is also used to set whether or not to display hidden file, file timestamps, file sizes and file attributes. The timestamps, file sizes, and file attributes are only visible as `TOOLTIPS` that display when the mouse cursor pauses over a file in the window.

- The `CodeFolio` tab presents a directory tree of pre-defined chunks of code, specific to various programming languages, that you can insert into your current document. You may also add your own Snippets to this directory. Refer to [CodeFolio Snippets](#) in [Chapter 6 Editing Features](#) for more information.

The Standard Toolbar

Hawk has several toolbars, but only the `Standard` toolbar is turned on by default during the initial Hawk session. Since the `Standard` toolbar is part of the default Hawk screen, it is introduced in this section; all of the toolbars are described in detail in [Chapter 12 Customizing the Interface](#).








Figure 2-7. The Standard Toolbar



The `Standard` toolbar buttons are explained below:


- The first nine buttons on the standard toolbar are for editing and file operations. These include creating, opening and saving files, printing, cutting, copying and pasting text, and redo/undo of the last change made.
- The next three buttons on the `Standard` toolbar relate to searching:
 - Use the `Toolbar Search Box` refers to the search capability built into the `Standard` toolbar in the form of a drop-down list box control. It provides the most immediate, convenient way to perform simple searches. The `Toolbar Search` honors all of the settings in the `Search Options` dialog.

To use the `Toolbar Search`, type in the string you wish to search for..

- Press  to repeat the last search.
- Press  to search for the word under the cursor in the current document.
- Press  to rebuild the current project.
- Press  to build the current project.
- Press  to go to the next message line.
- Press  to debug the module
- Press  to access Hawk's online `Help`.

3

Building Your First Project



This chapter provides an introductory guide for building Hawk projects and project spaces. Definitions of items in the `Project` menu are discussed more in [Chapter 7 Projects, Project Spaces, and Workspaces](#).

The sections covered in this chapter include the following:

[Projects and Project Spaces](#)

[Creating a Project Space](#)

[Creating and Modifying a Project](#)

Projects and Project Spaces

What are Hawk Projects and Project Spaces?

Hawk Projects offer a way to store related files as a unit. They make version control operations simpler and more convenient and they store unique configuration settings, such as compiler commands, with each project. There are no set criteria for deciding which files should go into a Hawk project.

Project spaces are an extension of the project facility. They store sets of projects as units, allowing multiple projects to be displayed at a time. Before a project can be made, a project space must be created.

The `New`, `Open`, and `Close` items on the `Project -> Project Space` submenu are for creating, opening and closing project spaces. Project spaces are displayed in the `File View` tab of the `Project Window`.

The appropriate steps for making projects are shown below.

1. Create a project space.
2. Make default settings.
3. Create projects and/or add existing projects to project space.
4. Add files to projects.

Creating a Project Space

To create a project space, complete the following steps:

1. Click `Project -> Project Space -> New`.
2. Click `Browse` to open the appropriate directory for your project space or enter the filename for the space and give it an extension.



The `.PSP` extension is automatically applied to the project space configuration file, which contains the necessary configuration information for the project space.

3. Click `OK`. The project space is created, and the `Project Properties` dialog appears, with the `Members` tab on top. The new project space is displayed in the list box on the left. Projects may be added to spaces and files are subsequently added to projects in this dialog.

However, before the project is created, it is necessary to set the project defaults; this is explained in the next section.

Setting Project Defaults

Configuration settings for new projects are inherited from the `<Default Settings>` item in Hawk's `Project Properties` dialog. This is the same dialog

that displays when you first open a new project space. It can also be accessed by clicking on **Project** -> **Settings**.

Changes made to the settings while a project is selected are stored with that project only. Closing the project causes all settings to revert to the defaults. For this reason, it may be a good idea to set some configurations for the project before creating it so the settings remain unchanged when the project is closed.

To make default settings throughout the **Project Properties** dialog, highlight the **<Default Settings>** item in the **Project** -> **Settings** dialog, then make various configuration settings in the various tabs of the dialog. Once you have established the default settings, you can make modifications for future projects on a per-project basis.

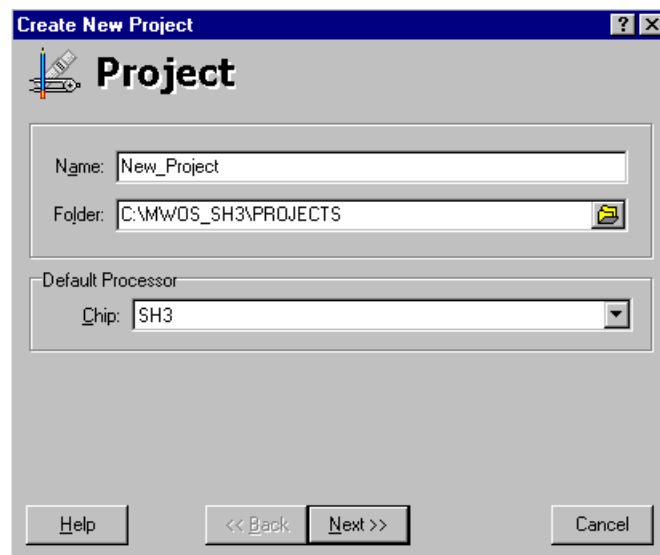
Creating and Modifying a Project

Once a project space has been created, individual projects can be created for the space. After creating a new project space, the **Project Properties** dialog appears. It is also possible to access the same dialog by clicking **Project** -> **Settings**.

However, we will not use the **Project Properties** dialog to add a project to the project space. The following steps will use a different method to create a Hawk project.

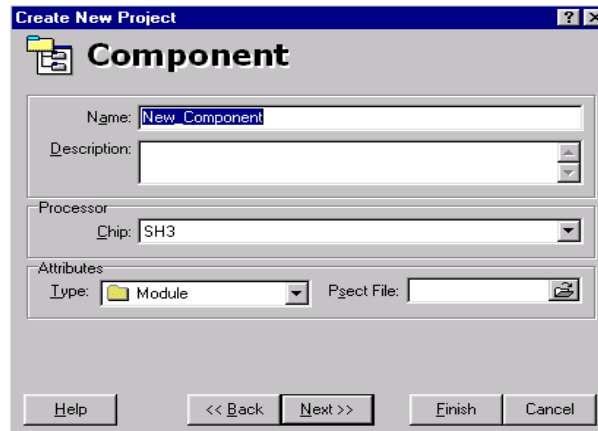
1. Click the **Cancel** button to dismiss the **Project Properties** dialog.
2. Click -> **Project Space** -> **Add New Project** to access the **Project** dialog. The dialog shown below appears:

Figure 3-1. Add New Project to Project Space



3. Enter the name of the project into the **Name** field. Also, make sure the path to the project's folder in the **Folder** field is correct and make sure the **Default Processor** displays the correct processor name.
4. Click **Next**. The **Component** dialog box appears, as shown below.

Figure 3-2. Add New Component to Project



- Projects consists of multiple components. Type the following into the component menu:

Name: **The name of your program/module**

Description: **Whatever description you want to include**

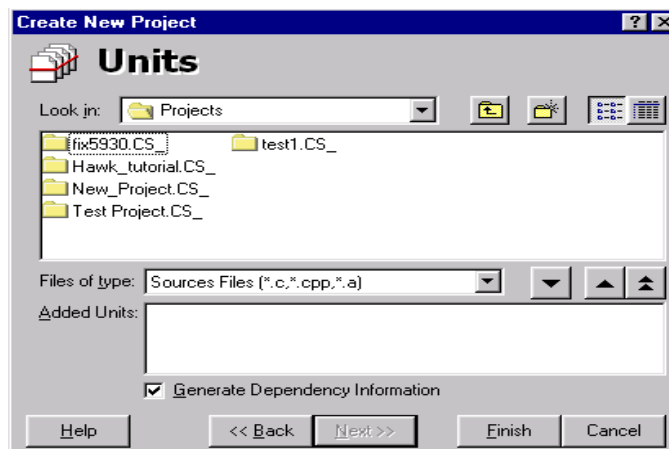
Chip: **<Processor Name>**

Type: **Type of module is specified here**

Psect File: Psect File can be left blank, Hawk will default to the correct psect file.

- At this point, you can do one of two things:
 - If you would like to add units to your project, proceed to step 6.
 - If you do not want to add units to your project, click **Finish** and proceed to step 11.
- Click **Next** to display the **Units** dialog. Components consists of units, which can be library, header, or source files. The **Units** dialog is shown below.

Figure 3-3. Units Dialog



- Navigate to the file you want to add from the **Look in:** box.



9. To add the file to the `Added Units` display, you can either click the down arrow to the right of the `Files of type` box or simply double-click on the file. The file you chose in the full path list should now appear in the `Added Units` box.
10. Repeat step 8 until you are through adding files. When you are done, click **Finish** at the bottom of the window. The `Components` frame should display a folder with the name of your added component.
11. To save the project, select **Project -> Save**.


Hawk creates a new project configuration file in the chosen directory that uses the name typed in the `Add New Project to Project Space` dialog with a `.PJT` extension appended. The new project is listed in the `Project Properties list box`, under the current project space.

Hawk creates a new configuration file in the chosen directory with the name that was typed and a `.PJT` extension.

Adding Files to a Project

Files can be added once the project has been created. Files are added in the `Project -> Settings -> Members` dialog. To add files to your project, complete the following steps:

1. Highlight a project to which the files will be added in the `Project Properties list box`.
2. Add files using one of the following four methods:
 - Click the **Add New File**  button.
 - Click the **Scan**  button. Add individual files by typing in their names, or add multiple files using file filters (such as `*.C`).

Within the `Scan` dialog, use the **Browse**  button to browse for files. Add individual files by typing in their names.

Matching files in subdirectories are included as a result of the `Include Subdirectories` check box in the `scan` dialog.



Semicolon delimited series of file specifications can be specified when adding files. These file specifications would normally include DOS wild-card characters. Paths are optional. (Examples include `*.C;*.H;*.CPP;C:\TEST*.C;C:\TEST*.H`).

- Use the `External Makefile` option to read files from an existing makefile into the project being created.
 - Use the `VCS Project` option to "read" a version-control project file. (See [Using Version Control in Hawk](#), in [Chapter 9 Version Control](#).)
3. Click **OK** when done. Any matched files should be immediately added to the project.


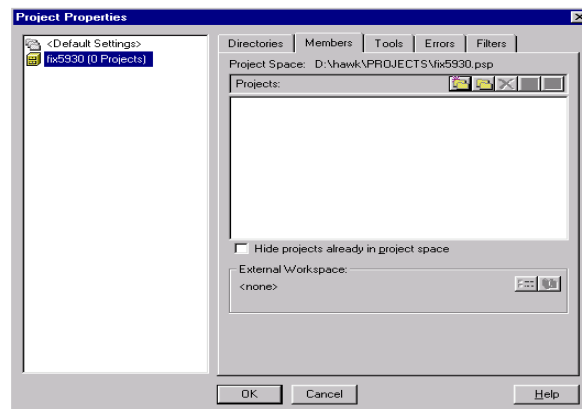
Files that are included in the project are listed in the `Files` list box in the center of the dialog. To remove files from the project, select the `Remove selected files from project`  radio button under the `Members` tab.

Figure 3-4. Project Properties Dialog: Members Tab

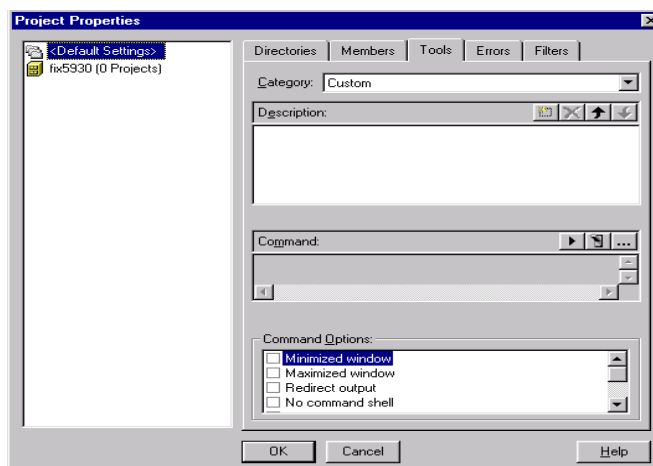


Project Tools

When the necessary files have been added in the `Members` tab of the `Project Properties` dialog, click on the `Tools` tab. The `Tools` tab is used for setting up various tools, such as command line compilers and version control utilities needed to perform various jobs on files being edited in Hawk.

The commands set on the `Tools` tab are the commands that are run when the various `Compile`, `Build`, `Debug`, etc., buttons and menu items are clicked. [Figure 3-5](#) shows the `Tools` Dialog.

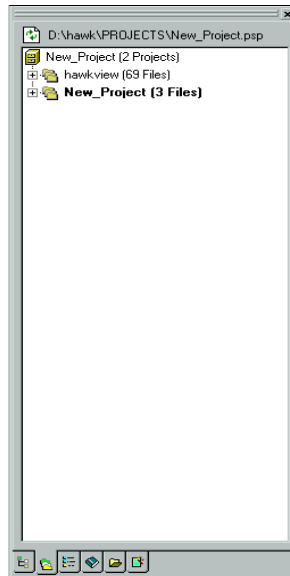
Figure 3-5. Project -> Settings -> Tools Dialog



Once the appropriate tools have been set on the `Tools` tab, click `OK`. The dialog closes and the project is created with the specified characteristics. Clicking on the project tab in the project window should now display the items in the project space.

Click on the **File View** tab of the **Project** window to display the project space with the included projects beneath. An example of this tab is shown in [Figure 3-6](#).

Figure 3-6. Project Window: File View



To load project files for editing, double-click on them in the window, or click on **Load Files** in the **Project** menu.

4

Command Key, Libraries, and General Environment

This following sections are included in this chapter:

[API Command Dialog/Prompt](#)

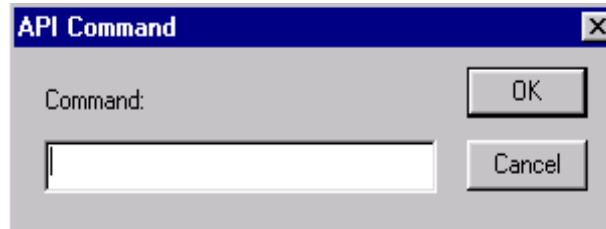
[Loading Hawk Add-Ons](#)

[General Environment Settings](#)

API Command Dialog/Prompt

The `API Command` item on Hawk's `Tools` menu interactively runs Hawk API functions. [Figure 4-1](#) shows the `API Command` prompt.

Figure 4-1. API Command Dialog



API functions drive the interface between the operating system and application programs. This includes the way in which the application communicates with the available operating system services. For example, APIs make it possible to open windows, display message boxes, and load dialogs.

Hawk has numerous API functions, many of which are available for interactive use within the Hawk editor. Hawk APIs can be used to change keystrokes, buttons, and menu commands. They can also be used in the macros and DLLs that extend Hawk as well as from the `API Command` prompt.

Use the `API Command` menu item to run Hawk's APIs on-the-fly from within Hawk. Simply type in the function name, with any necessary parameters, and press `OK`.

- To get help using any number of Hawk's APIs, access the *Functions* menu item from Hawk's online Help menu. Additional information can be found using the *Search for Help On...* menu item.

The `API Command` prompt can also be accessed with a special keystroke called the `Command Key`. The `API Command Key/Prompt` is described extensively in the following paragraphs.

API Command Key/Prompt

If you want to execute a command in Hawk and there is neither a key command nor a menu entry, you can use the `Command Key/Prompt`. Below are some guidelines to follow when using this prompt:

- If you are using the CUA or the vi keymap command set, the `Command Key` is `(.`
- If you are using the `BRIEF` compatible keymap command set, the `Command Key` is `)`; this also provides access to the `BRIEF` interpreter.
- In keymaps where there is no `KeyCommand` assignment, select the `API Command` from the `Tools` menu to access the prompt.

When you press the `Command Key` or choose `API Command` from the `Tools` menu, a prompt appears (as displayed in [Figure 4-1](#)) on the status line or in a pop-up dialog box, depending on which is enabled.

- You can respond to the prompt by entering a Hawk API function call, much as you would enter it in Hawk's C source code. Pressing **OK** sends the command off for processing (a list of some Hawk APIs can be found by accessing the `FUNCTIONS` menu item from Hawk's online Help menu).ab.



To specify a Status Line Command Key, choose the **Use Command Line Prompt** option on the `Customize -> Environment -> General` dialog under the `General` section. Refer to the topic *Command Line Response Editing*, in the online Help, for additional command line key sequences.

Command Completion

The `API Command` prompt has a command completion feature that completes function names using the first few characters of the function for reference. This feature is available for both the status line prompt and the pop-up dialog.

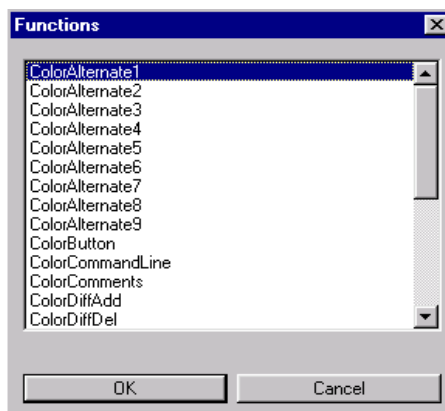
With command completion you can enter as much of a command (function name) as you want and press the **Tab** key to receive a list of all commands that begin with the same characters as the incomplete command.

Example

Complete the following steps:

- Select **Tools -> API Command**.
- Enter the word `color` and press **Tab** (shown in [Figure 4-2](#)).

Figure 4-2. Command Completion List After Entering "color"



- Select the desired command from the list to place it on the command line for you.



If you want a complete list of the commands available just press the **Tab** key without entering anything at the prompt. (Refer to Hawk's online help for additional help on any API function). More information about using Hawk's API and the `Command Key` can be found [Chapter 15 Extending Hawk](#).

Loading Hawk Add-Ons

The `Libraries` option on the `Customize` menu is used for interactively loading Hawk DLLs (Add-Ons) to extend the functionality of the Hawk program. A number of commonly used Add-Ons are contained in the list of Hawk Libraries in the `Libraries` dialog. The Hawk Libraries can be conveniently loaded by placing check marks next to the ones to be loaded.



DLLs are software used by Microsoft's Windows to provide services to applications. Some DLLs are automatically loaded when the program needs them, while others must be loaded at system startup. DLLs can be written in any programming language used for Windows programming. Three terms are used when referring to Hawk DLLs: Add-Ons, Libraries, and DLLs.

Loading a library in the `Customize -> Libraries` dialog adds a line similar to the following in the `CWRIGHT.INI` file and loads the DLL into Hawk (This is detailed in [Chapter 17 Configuration Files and Command Line Parameters](#)):

```
[LibPreload]
LibPreload=cwHTML.DLL
```

General Environment Settings

The `General` tab of the `Environment` dialog (accessed from the `Customize` menu) is used for setting various items in the Hawk environment. Items addressed by the settings in this dialog include the following:

- system prompts
 - Should system prompts (such as the `API Command` dialog/prompt) pop-up in a dialog or be displayed on the status line? Often, the preference is the status line prompt, the setting for which is `Use Command Line Prompt`. This is located under the `Customize -> Environment -> General` tab.
- windows and documents
 - Should windows and documents operate as a single unit, as they do in most Windows applications, or operate independently? The toggle for this functionality is `One Document Per Window`, on the `Customize -> Environment -> General` tab.
- recently loaded files
 - Should a list of recently loaded files appear at the bottom of the `File` menu? This option can be turned on by marking `Show File List on File Menu` under `Customize -> Environment -> File & File System`.
- Whether Unicode should be automatically detected and converted when files are opened and closed (mark `Auto-detect Unicode files`). Hawk does not


yet support Unicode internally. The Windows API function `WideCharToMultiByte()` is used to convert Unicode to ANSI Code Page characters when a file is loaded and then `MultiByteToWideChar()` is used to convert back to Unicode when the file is written out. You can also elect to convert some file types to/from Unicode from the `File -> Save As` dialog.

- window tiling
 - Should your window should tile vertically or horizontally, or in the traditional Windows way?

Visit the `Customize -> Environment -> General` dialog before beginning any editing tasks.

5

View Setups and Language Support



The `view setups` dialog contains attributes for color, fonts, line numbers, scroll bars, and other window color schemes. It also enables language support in Hawk. Thus, the following sections are included in this chapter:

[View Setups](#)

[Hawk Language Support](#)

View Setups

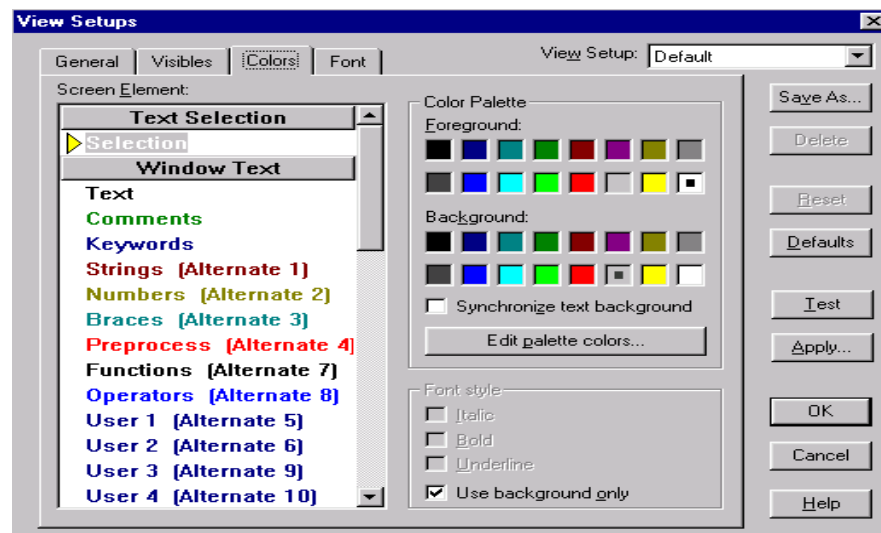
All items that pertain to Hawk's visual environment, such as colors, scroll bars, line numbers, and window settings, are part of Hawk view setups. View setups are controlled and manipulated in the `View Setups` dialog, which is accessed by clicking `Customize -> View Setups`.

There are four tabs on the `View Setups` dialog, including `General`, `Visibles`, `Colors`, and `Font`.

- The `General` tab is used for setting window attributes like scroll bars, line numbers, and margins. Use the `Highlight Current Line` option (in combination with the `Use Background Only` option on the `Colors` tab) to maintain font style and foreground colors.
- The `visibles` tab is used for manipulating "visibles"; this is a term used to describe what makes invisible characters, such as spaces and tabs, appear on the display. To turn the `visibles` option on, click `Visibles`. To give the `visibles` a slightly softer color, click on the `Use Visibles Color`. The main listbox lists the characters that are currently in the display. The characters can be changed using the combo-box located above the list of visibles.
- The `Colors` tab (depicted in [Figure 5-1](#)) is for changing screen colors. Colors can be changed for all text colors, such as the background, using the `Synchronize text background` option or selecting the element and then changing the colors (the latter option is for individual screen elements only).

Background and foreground palette colors can be changed into up to 16-million colors, assuming your display supports the colors. Several default view setups are provided to illustrate this feature (see the drop-down list in the `View Setup` field).

Figure 5-1. View Setups Dialog: Colors Tab



Any combination of changes (whether the changes be color and font, window and color, or visibles and color) made in the `View Setups` dialog can be saved with a

name clicking the `Save As` button and giving the view setup a new or existing name. Various view setups can be selected from the drop-down list under the `View Setup` combo box.

Any view setups can be restored to the settings that were made prior to entering the dialog, using the `Reset` button. The `Defaults` button returns only the `Default` and `Output View Setups` to their defaults. The `Test` button displays the latest changes to the current Hawk screen. The `Apply` button brings up the `Apply View Setup` dialog, which allows the selected view setup, with any changes, to be immediately applied to the `Current Window`, `All User Windows`, or to be made the `Default` view setup.

- The `Font` tab is for changing screen fonts. The fonts selected in the dialog do not affect printed documents. When using this tab, keep the following in mind:
 - Font styles can be combined.
 - The font style for individual screen elements can be changed, but only when the selected font supports the styles.
 - Many of the view setups, including `Default`, are set to use the 'Fixedsys' font. One benefit of this is that ChromaCoding font styles display upon first installation. Another benefit is that the extended umlaut (European) characters are supported by default.

Hawk View Setups

Hawk supplies several view setups out-of-the-box. They are listed for selection in the drop down list under the `View Setups` combo box. Many supplied View Setups have "16-bit" as part of their names. This means more than 256 colors are needed to reproduce the palettes (16-bit color is the same as 65536 colors).

- Among the list of view setups, the ones named `Default` and `Output Window` are of particular interest. Both of these view setups affect windows in a particular way and have additional screen elements available for modification.



IScreen elements are the elements on the Hawk screen whose colors can be changed. Examples of screen elements are functions and keywords for supported programming languages. Screen elements are listed in the `Screen Element List` in the `Colors` section of the `View Setups` dialog.

- If the `Default` view setup is selected, additional screen elements will be available for controlling colors and fonts of system elements such as the status bar. The items are in the `System` group at the bottom of the list.
- If the `Output Window View Setup` is selected, additional items are available for controlling window attributes of the `Output` window.

Default View Setup

The `Default` view setup is the view setup that is initially associated with all Hawk windows. Changes made to this view setup affect all windows that do not have view

setups otherwise associated with them. The two ways in which you can use another view setup as the default include the following:

- Use the **Save As** button in the **View Setups** dialog to save the selected view setup as **Default**.
- Mark **Make Default** in the **Apply View Setup** dialog while the desired view setup is selected in the **View Setups** dialog.

Output Window View Setup

Changing the **Output** window view setup changes the colors of screen elements in the **Output** window. To change window settings for individual tabs of the **Output** window, make the changes, then save the view setup as the following:

```
"Output Window n"
```

(Where *n* stands for the number, 0 - 9, of the tab being changed)

Using View Setups

As mentioned, windows in Hawk are initially associated with the **Default** view setup. To use any other view setup, it must be associated with a Hawk window. With this in mind, consider the following to change the colors and window attributes of a window or set of windows:

- Change the options for the **Default** view setup in the **View Setups** dialog in order to change the colors, fonts, and other window attributes of a window associated with the **Default** view setup.
- Associate any view setup with the currently open window by selecting the view setup and then marking **Current Window** in the **Apply View Setup** dialog (accessed by clicking **Apply** in the **View Setups** dialog).
- Associate any view setup with selected open windows by choosing the view setup for the selected windows in the **Document/Window Manager** dialog.
- Associate any view setup with all open windows by marking **All User Windows** in the **Apply View Setup** dialog (accessed by clicking **Apply** in the **View Setups** dialog).
- Associate any view setup with closed or open windows containing files of a certain file type, such as **.C**, **.H**, **.CPP**, by choosing the view setup for the selected file type in **Customize -> Language -> Coloring**.

Hawk Language Support

Hawk's language support consists primarily of the following elements:

- ChromaCoding
- Template Expansion
- Smart Indenting

Languages that are supported out-of-the-box include file-types with the following extensions: **.C**, **.CPP**, **.JAVA**, **.HTML**, and **.ASM**. There are also some languages for

which language Add-Ons are available, but not loaded. In these cases, a procedure for loading the Add-Ons is required to enable the support.

Ultimately, through the creation of ChromaCoding Lexers and language Add-Ons, support can be enabled for any programming language. The following topics discuss lexers and Add-Ons and the various ways they are configured to provide support for languages not immediately supported by Hawk out-of-the-box.

Language Support Lexers and DLL's

There are two ways that Hawk provides language support functionality-- through ChromaCoding Lexers and DLLs; in order to have language support features such as ChromaCoding and Template Expansion, there must be a lexer or a DLL.

Some languages are supported by default. For example, when files with `.C` extensions are opened in Hawk, ChromaCoding is immediately apparent and template expansion can be enabled and used by marking a single option in the **Customize -> Language** dialog. In this sense, files with `.C` file-type extensions are supported by Hawk out-of-the-box. When a language is supported out-of-the-box it often gets its functionality from Hawk's `CWSTART.DLL`, which loads automatically each time Hawk is launched.

There are two options for language support not loaded out-of-the-box. They include the following:

- Support can be obtained from external Add-Ons (DLLs), which must be loaded manually.
- OR -
- Support can be obtained from a `Lexer`, which can be created and/or modified in the `ChromaCoding Lexer Settings` dialog.

The following topics describe the two methods for adding language support to Hawk. When reading the information, keep in mind that ChromaCoding Lexers are the easiest way to create new language support for Hawk. ChromaCoding provided by lexers is also faster, which means the color display is smoother.

In addition, lexers do not support Smart Indenting or Template Expansion at this time. These features are only provided with language Add-Ons. For this reason, it is best to use Add-Ons with lexers (if available). In such cases, the lexer's functionality takes precedence only when the same functionality (such as ChromaCoding) is provided by both.

Language DLLs

When language support is obtained from a DLL, the first thing to do is make sure that the DLL for the respective language is in the Hawk directory. Some language support DLLs are already in the Hawk directory. Many more can be downloaded from Starbase's web pages. On the web pages, the support comes in the form of a zip file, which contains the supporting DLL, some source code, and, at times, a keyword file and a `README` file.

The Tools Dialog

After making sure that the appropriate DLL is in the Hawk directory, you need to load the library. Hawk libraries (including language support) are loaded using Hawk's `Customize -> Libraries` dialog. The DLLs will add functionality to Hawk, whether that functionality is language support or another feature.

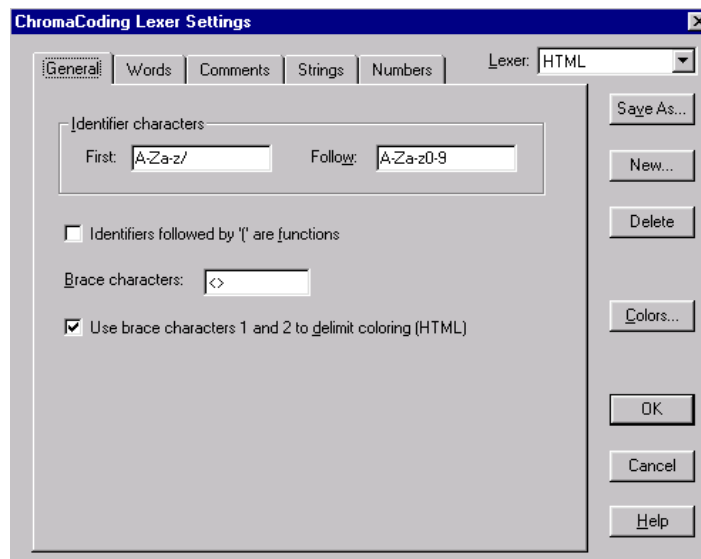
Some Add-Ons are listed in the `Libraries` dialog in the list of `Hawk Libraries`. If a language is not supported out-of-the-box, search for the language in the list of `Hawk Libraries`. The library can be loaded by clicking `Add` and browsing for the appropriate DLL. For more information about the `Libraries` dialog, see the [Loading Hawk Add-Ons](#) section in [Chapter 4 Command Key, Libraries, and General Environment](#).

ChromaCoding Lexers

ChromaCoding Lexers provide language support, such as ChromaCoding, for programming languages edited in Hawk. They store collections of settings to color various elements of the programming language's "vocabulary". These vocabulary elements include identifiers, braces, preprocessors, keywords, operators, strings and comments. Lexers are the easiest way to add support for a language that Hawk has not previously supported.

ChromaCoding Lexers can be created or modified in the `ChromaCoding Lexer Settings` dialog (shown in [Figure 5-2](#)).

Figure 5-2. ChromaCoding Lexer Settings Dialog



There are two ways to access the `ChromaCoding Lexer Settings` dialog:

- Click `Customize -> ChromaCoding Lexers...`

-OR-

- Click `Customize -> Language -> Coloring` to get the `Language` dialog box. Then click `Settings` under the lexer option button.

Creating a Lexer

The following sections provide steps for creating a lexer:

- [Make a Lexer Name](#)
- [Define Identifier Characters](#)
- [Define Keywords](#)
- [Define Comments](#)
- [Define String Delimiters](#)
- [Define Numbers](#)

Make a Lexer Name

To make a Lexer Name complete the following:

1. In the `ChromaCoding Lexer Settings` dialog, click **New**.
2. Type a descriptive name for the lexer.
3. Click **OK**.

A new Lexer should appear in the `Lexer`: combo-box.

Define Identifier Characters

Enter identifier characters for the language on the `General` tab of the `ChromaCoding Lexer Settings` dialog.

Identifier characters are characters that distinguish "identifiers" from other elements of the language. Identifiers can best be described as the elements of the language that make up the "words" of the language. Keywords and function names are identifiers, but operators are not. If identifier characters are not defined, ChromaCoding for identifiers will not work.

Character classes (described under the topic [Regular Expressions](#) in [Chapter 10 Search and Replace and Other Navigational Tools](#)) can be used when defining identifier characters.

Examples

- `A-Za-z$`
`_A-Za-z0-9$`



Certain regular expression characters must be escaped with a back slash (`\`) in order to use them literally. For example, to specify a dash (`-`) character as an identifier character, enter it as `\-`.

There are two fields in the `Identifier Characters` section of the `ChromaCoding Lexer Settings` dialog. They include the following:

- `First` is for characters that are allowed to appear as the first character of the Identifier.

- Follow is for characters that are allowed to follow the first character of each Identifier.

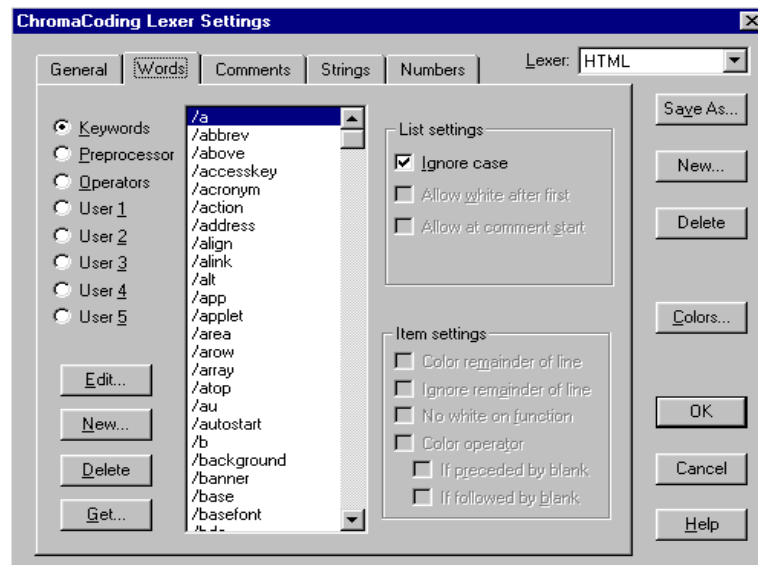
The options on the lower half of the General tab apply only to certain languages and indicate that identifiers followed by parentheses are functions (like C++). In addition, the options address the following questions:

- If braces are used, which should be colored?
- Should the first two braces defined delimit keywords, numbers, and strings for coloring?

Define Keywords

On the Words tab of the ChromaCoding Lexer Settings dialog (shown in Figure 5-3), add any keywords, operators (such as +, -, *, &), or preprocessor words (if appropriate) that the ChromaCoding Lexer should color.

Figure 5-3. ChromaCoding Lexer Settings -> Words Tab



Use the following steps to add words:

1. Choose the radio button for the group (such as Keywords or Operators) to which the items will be added.
2. Click the **New** button on the left.

3. In the resulting `Add Words` dialog, type the appropriate character(s) to add.



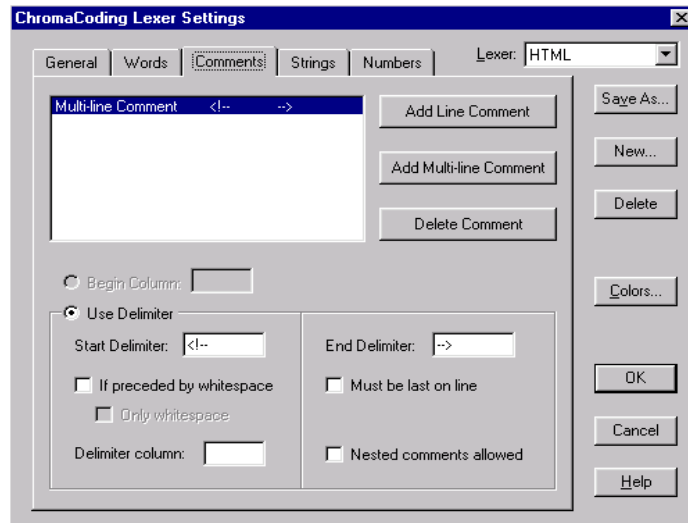
- Items can be deleted from a list by selecting the item and pressing the `Delete` button to the left.
- The `Get` button on the bottom left adds items from a file. Clicking it brings up the `Get Words from File` dialog, which is used to browse for the file containing the items to be added. The `Get Words from File` feature can only read words from a file that are separated by white space. It can not differentiate words from other characters such as parentheses or colons.
- Added items will display in the center list-box.
- The `Edit` button can be used to modify one or more words in a list.
- "User..." items are for user-defined keywords, which can be colored differently than the `Keywords` items (Colors are set in the `View Setups` dialog, described in the previous topic on [View Setups](#)).
- The `List Settings` section on the top right has items that apply to a list of words that are currently displayed. Certain items apply only to certain sets. For example, `Allow White After First` (which causes preprocessors to color even if whitespace appears after the first character) applies to lists of `Preprocessors` only.
- The `Item Settings` section has options that only apply to individually selected items.
- Certain check boxes on the `Words` tab apply only to certain sets. For example, `Ignore Remainder of Line` is only available for the list of `Preprocessor` keywords. This option tells the Lexer not to ChromaCode any text after the selected preprocessor. It is useful for languages like C++ that have preprocessor lines similar to the following:
 - `#include "exports.h"`
 - where `"exports.h"` would otherwise be incorrectly colored as a string

Refer to Hawk's online help for additional information on the options available on the `Words` tab.

Define Comments

- On the `Comments` tab (identified in [Figure 5-4](#)) define all the appropriate comment delimiters for each comment set. The configurations then tell the ChromaCoding Lexer when to color comments.

Figure 5-4. ChromaCoding Lexer Settings -> Comments Tab



- To define single line comment delimiters, click the **Add Line Comment** button. To define multi-line comment delimiters, click **Add Multi-line Comment**. Each time either of the buttons is clicked, the **Begin Column** or **Use Delimiter** options become available, depending on which radio button is chosen at the time.
- If a single-line comment is being added, the right side of the **Use Delimiter** section is not available.
- If a multi-line comment is being added, the **Begin Column** option will not be available.
- Finish setting up the comment support by completing the fields and command options that apply to the characteristics of the comments for the language in question.

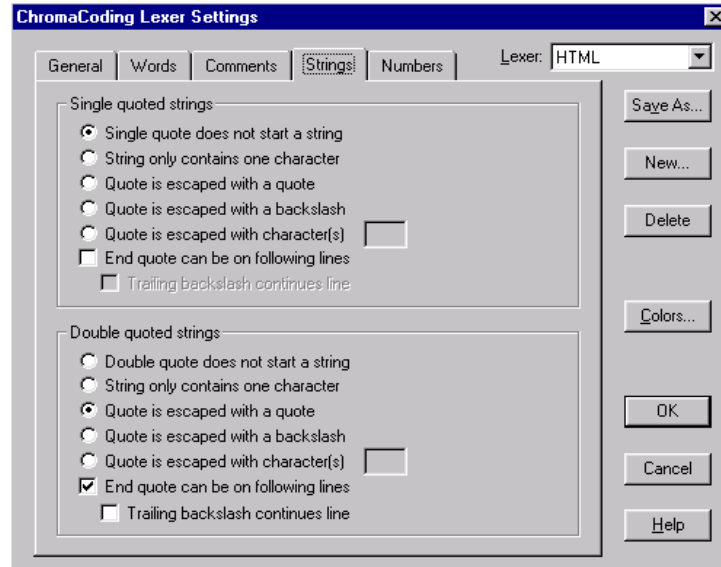


More information on the options available in the **Comments** tab can be found in Hawk's online help.

Define String Delimiters

On the **Strings** tab of the **ChromaCoding Lexer Settings** dialog (displayed in [Figure 5-5](#)), choose the options necessary to tell the ChromaCoding Lexer how to color single- or double-quoted string delimiters. If the language in question does not use quoted strings, mark the option(s) specifying the following: **Single/Double quote does not start a string**.

Figure 5-5. Strings Tab



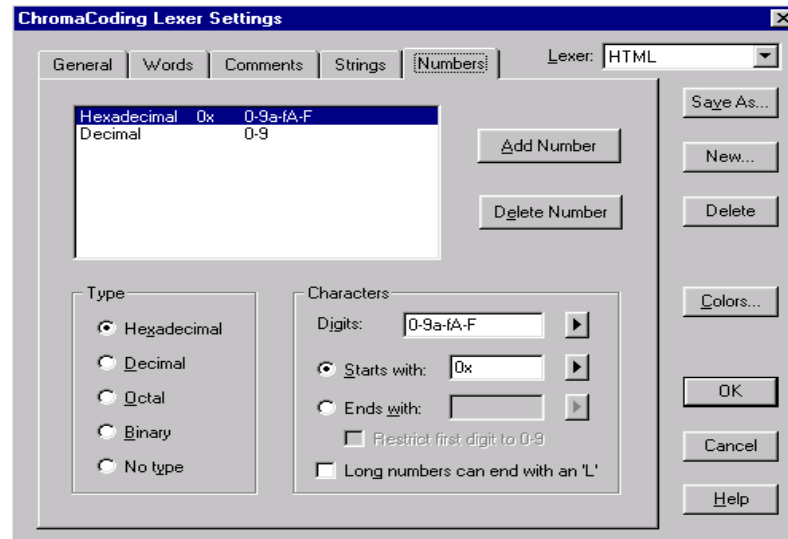
Define Numbers

On the `Numbers` tab (shown in [Figure 5-6](#)), define any numbers that should be colored by the ChromaCoding Lexer for the language in question. The available number types include the following:

- Hexadecimal
- Decimal
- Octal
- Binary
- No type

The number types only serve to identify an entry. They are there for convenience. When one of the types is chosen, the most appropriate number set is automatically inserted in the `Digits` field. The name of the chosen type is also displayed, for reference purposes, beside the number set in the `Numbers List Box`. The other settings in the dialog control how the elements are ChromaCoded by the lexer.


Figure 5-6. Numbers Tab



Use the following steps to define number elements:

1. Click **Add Number**.
2. Choose the number type to be applied. A new item will appear in the list box at the top of the dialog.
3. Add the numbers to the **Digits** field.

Character classes (such as 0-9, indicating all numbers between 0 and 9) can be used.

Commonly used character classes for the selected number type can be inserted using the pop-up menu with the black right-arrow button ; this button is to the right of the **Digits** field.

Starts With/Ends With

Numbers that are used in programming languages are often delimited with specific starting or ending characters. For each added set of numbers, beginning and ending characters can be specified for the ChromaCoding Lexer using the **Starts With** and **Ends With** fields. These fields also have black right-arrow buttons to access pop-up menus containing commonly used beginning and ending characters. The **Restrict first digit to 0-9** option is available when using the **Ends With** field. It restricts the lexer to color numbers that end with certain characters only if the first digit is numerical.

The last option in the **Lexers** dialog, **Long numbers can end with L**, specifies that ChromaCoding includes the **L** when coloring languages that end long numbers with **L**.

Completing the Lexer

- When all of the appropriate language characteristics have been added to the Lexer, click **OK** for the dialog. If you want to delete a lexer, choose it from the drop-down

list under the Lexer combo-box at the top of the ChromaCoding Lexer Settings dialog. Click the **Delete** button.



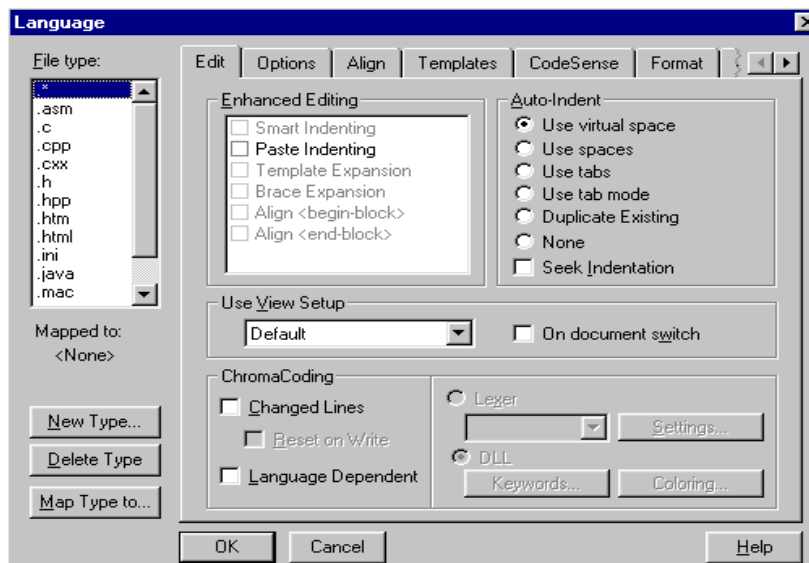
All changes made in the ChromaCoding Lexer Settings dialog can be cancelled by clicking the **Cancel** button.

Configuring Options in the Language Dialog

Whether language support is obtained from a language DLL or a ChromaCoding Lexer, the configurations for the language need to be set up. This is done in the Language dialog, accessed by selecting the **Customize** -> **Language** menu options.

The Language dialog (shown in Figure 5-7) is where all Hawk's language features are stored and controlled. Hawk's language features consist of settings and support associated with file type extensions (such as .C, .TXT, .ASM). The available features vary depending on the extension of the file.

Figure 5-7. Customize -> Language



Some of the options in the Language dialog are available for any file type; others are only available with file types for which a language support module is available. For example, the **Paste Indenting** feature can be used for any file type, while **Smart Indenting** features (described [Chapter 6 Editing Features](#)) are only available for some file types. If a language support module does not provide support for a particular feature in the dialog, the feature will not be available.

The Language dialog has seven tabs, which are, along with other features, described below. When making changes in the Language dialog, remember that the changes are file-type specific, that is, they apply only to files with extensions highlighted in the **File Type** list at the time of the change.

File Type List

The `File Type` list on the left-hand side of the `Language` dialog lists all of the available file types for Hawk. This dialog is displayed in [Figure 5-8](#).

Figure 5-8. File Type List



A file type is the extension of a given file (such as `.c`). It is used for selecting the file type whose settings will be viewed or modified. The file type associated with the current document is selected initially, if it is on the list. If it is not on the list, it can be added, or the default settings (indicated by file type `.*`) apply. To select more than one extension at a time, use `Ctrl` or `Shift`.

Options Tab

The `Customize -> Language -> Options` dialog has options for controlling certain attributes and functions for files of the selected type. Some of the options available are as follows:

- `Read-only` controls whether files of the selected type should be opened as read-only.
- `Make backups` controls whether backup files should be made for files of the selected type.
- `EOL Options` control whether UNIX, DOS, or Mac EOL characters should be inserted when the `E` key is pressed in files of the selected type.
- `Word Wrap` options control whether and how text should be wrapped in files of the selected type.

Most of the features in the `Language -> Options` dialog are also available on a per-document, non-file-type-specific basis, in the `Document/Window Manager` dialog.

Tabs/Indenting Tab

The `Tabs -> Indenting` tab of the `Language` dialog has options for controlling and customizing how Hawk handles tabs, virtual space, and auto-indenting. It also offers a choice of several popular forms of `block Alignment Styles` which are used when

inserting templates and/or when using brace expansion. The available alignment styles are:

- Unindented Block
- Line Saver
- Indented Block

Most of the features in the `Language -> Tabs/Indenting` dialog are also available on a per-document, non-file-type-specific basis, in the `Document/Window Manager` dialog.

Templates Tab

- The `Templates` tab of the `Language` dialog is for examining and redefining language templates associated with the selected extension.



More information about templates is available in [Chapter 6 Editing Features](#).

Coloring Tab

The `Coloring` tab of the `Language` dialog has options for controlling how `ChromaCoding` operates. It is also where `Hawk` is configured to provide `ChromaCoding` support for languages that are embedded in other languages. The items in the dialog that are of special interest are the `Lexer` and `DLL` options, and the `Embedded Languages and Scripts` section.

- Mark the `Lexer` radio button to make the `Lexer` options available and to cause the combo-box to display the currently selected `Lexer`, as described in the section on *ChromaCoding Lexers*. `ChromaCoding` lexers maintain numerous settings that add support to and for programming languages edited in `Hawk`. To choose a different lexer, select it from the drop-down list under the combo. The lexer displayed in the combo box will be associated with the selected file type in the `File Type` list.

The `Settings` button can be used to access the `ChromaCoding Lexer Settings` dialog, used for making new `ChromaCoding Lexers`. The dialog can also be accessed by clicking the `ChromaCoding Lexers` option on the `Customize` menu.

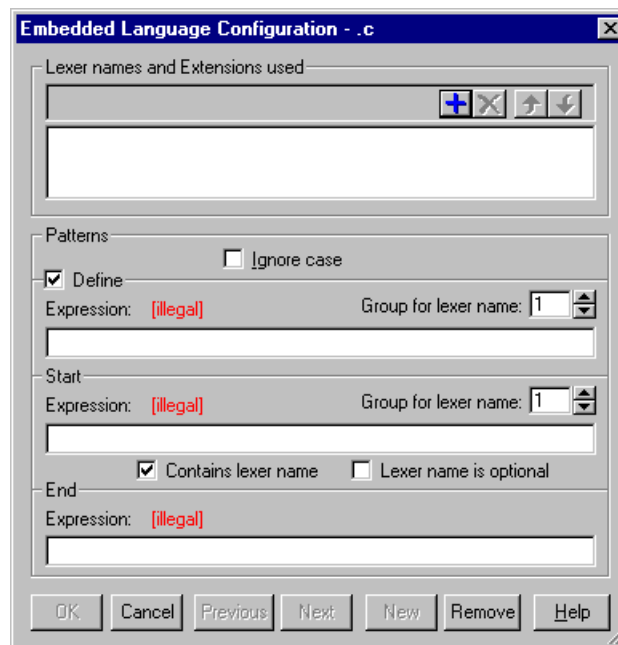
- Mark the `DLL` radio button to have `Hawk` use a `DLL`, rather than a lexer, for language support. Marking the `DLL` option enables two additional buttons.
 - The `Keywords` button accesses the `DLL Extension Keywords` dialog, used for defining and editing a file of keywords to supplement the ones predefined by the `DLL`.
 - The `Coloring` button accesses the `DLL Coloring` dialog, used for controlling `ChromaCoding` color-timing and the number of comment lines that will be colored. If there is no `DLL` loaded for the selected file type, the `DLL` radio button will be unavailable.

These buttons are not available when the `Lexer` option is chosen.

- The `Embedded Languages and Scripts` section has the following components and rules:
 - The `Configure` button accesses the `Embedded Language Configuration` dialog. The button will not be available if the selected file type does not use a `ChromaCoding` lexer for `ChromaCoding` support.

The `Embedded Language Configuration` dialog uses regular expressions to parse programming languages that are embedded in other programming languages (e.g. JavaScript in an HTML file). It then colors the embedded language using a designated `ChromaCoding` lexer. Other language support features, such as template expansion, are also provided. With the proper configurations, Hawk will be able to distinguish the embedded language from the main language, and provide support for both.

Figure 5-9. Embedded Language Configuration dialog



Help for the dialog is available online, under the topic *Embedded Language Configuration*. See the topic [Regular Expressions](#) in [Chapter 10 Search and Replace and Other Navigational Tools](#) for help with regular expressions.

- `<not defined>` will appear if no embedded languages have been defined for the file type. `<n defined>` will appear if one or more embedded languages have been defined, where n is the number of defined languages.
- There must be an existing `ChromaCoding` lexer that provides color support for the embedded language in order for the embedded language configuration to work. If an appropriate lexer does not exist, it can be created in the `ChromaCoding Lexer Settings` dialog.

The option `Use 'background override' color` in `Customize -> Language -> Coloring` allows different background colors to be set for blocks of embedded code. It uses the colors that are defined for the screen element

'Background Override' in `Customize -> View Setups`. Use 'background override' color is file type specific and should be marked for the file type of the parent language.

CodeSense Tab

- The `CodeSense` tab of the `Language` dialog is used for selecting and manipulating the `Parser Selection` and `Name Completion`. These features parse specified text from files being edited to use for editing or navigational purposes.



More information about `Name Completion` and `CodeSense` can be found in [Chapter 6 Editing Features](#). Symbols are described in [Chapter 10 Search and Replace and Other Navigational Tools](#).

Format Tab

The `Format` tab of the `Language` dialog contains options for `Code Reformatting` or beautifying. These options are used by the `Format Source` item on the `Text` menu. `Format Source` is explained in detail in [Chapter 11 Checking and Formatting Files](#).

Comments Tab

The `Comments` tab of the `Language` dialog is used for setting the parameters for the `Comment` and `Comment Box` menu items on the `Text` menu. The `Comment` features are explained in more detail in [Chapter 6 Editing Features](#).

Adding a New File Type to the Language Dialog

Loading a language support `DLL` or making a `Lexer` does not mean that `Hawk` will automatically recognize and color the file type of the file for which the support was added. To associate language-support to a file type, the file type must first be added to the `File Type` list in the `Language` dialog. Complete the following steps to add a new file type in the `Language` dialog:

1. Select `New Type` and add the extension for the language (without the '!', which is automatically added). Once the file type has been added, select it in the list of file types on the left.
2. In the same dialog, select the `Coloring` tab.
 - If the file type will be getting its `ChromaCoding` support from a `Lexer`, select the `Lexer` radio button and choose the appropriate `Lexer` from the `Lexer` combo-box to associate it with the selected file type.
 - If the file type will be getting its `ChromaCoding` support from a `DLL`, mark the `DLL` radio button.
3. Once the language support method has been established, check and configure the options on `Tabs/Indenting` tab (these features will be covered in [Chapter 6 Editing Features](#)) and on the `Coloring` tab to enable language-specific editing features and coloring for the file type.

When a source file with the new extension is opened, syntax colors (ChromaCoding) should be visible. Other features supported by the corresponding Add-On or lexer are also available.

Aliasing

If there is no language support DLL for a particular file type, there are two remaining options available, including creating a DLL or adding support by creating a lexer (described in the previous section). There is some information on creating a language DLL in [Chapter 15 Extending Hawk](#).

An alternative way to add language support is to alias the unsupported file type with a supported file type. If the code style of the unsupported language is similar to one that is supported, all of the templates, keywords, and coloring for the existing file type are used by the unsupported file type. Keywords can be added to the supported file type's list for additional coloring.

To alias, or map, a file type to one that already exists, complete the following steps:


1. Go to the **Customize -> Language** dialog and press the **New Type** button.
2. Add the **_new_** file type.
3. Highlight the **_new_** file type in the list of file types on the left.
4. Press the **Map Type to** button and enter the file type that will be the alias for the unsupported language.
5. Save and exit the dialog.

When source files of the unsupported extension are opened, they use the language support of the file type to which they are mapped.

Some of the tabs in the **Language** dialog are unavailable for mapped or aliased files. This is due to the fact that aliased file types use the language-specific functions of the type to which they are mapped. In such cases, changes need to be made to language-specific features (such as template expansion) for mapped file types can only be made to the original file type to which the unsupported language is aliased.

6

Editing Features



This chapter covers the following sections:

Templates and Brace Expansion

Indenting

CodeSense

Comments and Comment Boxes

HTML Editing

Hex Editing

Clipboard and Scrap Buffers

API Assistant

Using Help in Hawk

Printing

Templates and Brace Expansion

This section discusses working with templates and brace expansion.

Templates

Templates may be language-specific (associated with a particular programming language) or language-independent. They make it possible to insert frequently used text into various files at the press of a button. In addition, when appropriate abbreviations are used, templates can automatically expand common statements made in different programming languages.

Language-Specific Templates

Language-specific Template Expansion is activated on a per-file-type basis in the **Customize -> Language -> Templates** dialog. To use it, highlight the appropriate file type and mark the **Template Expansion** option.

Templates work similarly to the example shown below.

Example

Type `if` and press the space bar.

The appropriate ending statement and any necessary parentheses or curly braces are inserted in the appropriate positions for the statement, as shown below:

```
if ( )
```

Creating and Modifying Language-Specific Templates

Because templates are associated with a specific programming language, Hawk associates them with file types used in that programming language.

Two methods are provided for adding template constructs or changing those already in existence:

- Templates defined for a particular file type can be viewed, added, and changed on the **Customize -> Language -> Templates** dialog. Use the dialog to view, create, and modify the abbreviations that trigger template expansion and the string values associated with them.
- Add templates by directly editing Hawk's configuration file. Use a call to `ExtAssignTemplate` under the [Editor] heading of the `CWRIGHT.INI` file. `ExtAssignTemplate` is a Hawk API.
- There are three string parameters used by `ExtAssignTemplate`. The first discusses which file extension to associate with the template (such as `.c` or `.sc`). The second string parameter specifies which word or abbreviation should trigger the template expansion (such as `"if"` or `"else"`). The third string is the template.



Help for Hawk APIs can be found in Hawk's online help using the [Search For Help On...](#) menu item in [Help](#) menu.



More information on `CWRIGHT.INI` is provided in [Chapter 17 Configuration Files and Command Line Parameters](#).

Special characters exist for use in language templates. [Table 6-1](#) defines these characters.

Table 6-1. Special Characters

Character	Purpose
<code>\n</code>	New Line--simulates pressing Enter
<code>&</code>	Specifies cursor position after template insertion
<code>@</code>	Issues a backspace
<code>\c</code>	Insert 'c' literally, (for instance, <code>\&</code> , <code>\@</code> , <code>\\</code>)
<code>\t</code>	Insert a tab

Examples

Most templates take up more than one line in the document. Therefore, you need to specify the locations of new lines in the template string. In most templates, you also need to specify a location in which the cursor is placed after the template is inserted.

Below are some sample template strings that show how special characters are used to construct templates:

```
"if (&)\n{\n}"
"do \n{\n\t&\n}\n while( );"
"for (& ; )\n{\n}"
```

The following line shows how the complete `ExtAssignTemplate` line might look in the configuration file:

```
ExtAssignTemplate=".PAS","proc","Procedure &();\nBegin\nEnd;"
```

The above example adds a template to use when editing files with the extension `.PAS`. When the word "proc" is typed and followed by a space, a template for a Pascal Procedure is inserted in the document. The cursor is positioned at the point where the name of the procedure would be entered.

The `ExtAssignTemplate` function is described in the Hawk's online help.

Non-Language-Specific Templates, Function and File Headers, and Macros in Templates

The templates described up to this point have been language-specific, which means they need language support to work. For more information about this topic, see [Chapter 5 View Setups and Language Support](#).

Templates that can be used without language support are described in this section, as well as template macros (macros that perform specific functions inside templates). Note that template macros can be used in language specific macros, as well as non-language-specific macros.

Two templates provided with Hawk are prime examples of non-language-specific templates. These templates insert C++ function and file headers into the current document. Even though the headers contain C++ language constructs, they can be modified. In addition, new headers can be created that will insert any kind of text, whether it is for a programming language or not. The headers are contained in two template files in the Hawk home directory. The names of the files are `FILE.TPL` and `FUNCT.TPL`.

- Two buttons on the `Edit` toolbar are used to insert the contents of `FILE.TPL` and `FUNCT.TPL` into the current document. The buttons use the function `ExtExpandTemplate` to insert the templates. The same function can be used to insert templates with any Hawk button or Keystroke. Details for the `ExtExpandTemplate` function can be found in Hawk's online help..



The `Edit` toolbar is turned on by marking the `Visible` option in the list of toolbars on the `Toolbars` tab of the `Tools -> Customize -> Toolbars` dialog.

To see examples of how `ExtExpandTemplate` is used, look at the function bindings for the two buttons on the `Edit` toolbar. The function bindings for the toolbars can be viewed on the `Bindings` tab of the `Toolbars` dialog. To view the function bindings, highlight the `Edit` item in the list of toolbars and scroll through the toolbar's buttons in the right window. The function bindings for each button will appear in the `Function Bindings`. The first two buttons on the `Edit` toolbar are the buttons that insert the headers.

The function binding for the function header looks similar to the following:

```
ExtExpandTemplate %ffunct.tpl$
```

This command uses the `%f` template macro to insert the contents of `FUNCT.TPL`, which contains the function header. The trailing `$` is used to delimit the filename string.

Template macros can also be used within templates. `FUNCT.TPL` contains the `'%q'` macro, which presents a prompt that queries the user for information which will be used in the template. To understand how template macros can be used, a few more macros will be added to `FUNCT.TPL`.

Below are the contents of `FUNCT.TPL` without modifications:

```
/*
** %qEnter function name:$
*
*   PARAMETERS:
*
*   DESCRIPTION:
*
*   RETURNS:
*/
```

The second line contains the `%q` query macro, which requests the name of the function.

Below is a modified version of the same file:

```
/%rep*60
```

```

** %qEnter function name:$
*
* PARAMETERS: &
*
* DESCRIPTION:
*
* RETURNS:
*
* CREATED: %date %time
*
* BY: %eUSERNAME$
*/

```

Below are the added macros:

`%rep` This macro repeats the `*` character 60 times.

`&` This macro specifies where to place the cursor at the end of the insertion.

`%date` This macro inserts the current date.

`%time` This macro inserts the current time.

`%e` This macro inserts the value associated with the specified environment variable, `USERNAME`. The `$` character is used to delimit the string.

Below is an example of a header added by the expansion of this template file:

```

/*****
** MyFunc
*
* PARAMETERS:
*
* DESCRIPTION:
*
* RETURNS:
*
* CREATED: 08/16/95 11:45:50
*
* BY: milow
*

```

- A complete list of the `%` macros available is displayed in [Table 6-2](#).



The function and file headers and template macros described in this section can also be used and inserted with Hawk's CodeFolio Snippets feature, described in the section on [CodeFolio Snippets](#).

Table 6-2. Macros in Templates

% Macro Form	Description
<code>%colNum</code>	Move the cursor to column <i>Num</i> of the current line.
<code>%date</code>	Insert a U.S. formatted date string at the current cursor position (mm/dd/yy).

Table 6-2. Macros in Templates

<code>%db</code>	Delete to the beginning of the line.
<code>%dcNum</code>	Delete <i>Num</i> characters at cursor.
<code>%de</code>	Delete to the end of the line.
<code>%dlNum</code>	Delete <i>Num</i> lines, beginning with the current.
<code>%dw</code>	Delete the word at the cursor.
<code>%eEnVar\$</code>	Insert the string value associated with environment variable, <i>EnVar</i> .
<code>%fFilename\$</code>	Insert the named file at the cursor. Any template macros within the file are also processed.
<code>%home</code>	Move the cursor to the beginning of the line.
<code>%lineNum</code>	Move the cursor to the line named by <i>Num</i> .
<code>%mdNum</code>	Move down <i>Num</i> lines.
<code>%meof</code>	Move the cursor to the end of the file.
<code>%meol</code>	Move the cursor to the end of the line.
<code>%mlNum</code>	Move the cursor left by <i>Num</i> columns.
<code>%mrNum</code>	Move the cursor right by <i>Num</i> columns.
<code>%muNum</code>	Move up <i>Num</i> lines.
<code>%Num</code>	See <code>ExtSetTemplateMacro</code> . When <i>Num</i> is 0 to 9, this refers to a user-definable string that may be different for each extension. Any macros contained in these strings are also expanded. These definitions are normally stored in your configuration file or <code>CWRIGHT.EXT</code> . The macros 0 through 3 are used for custom indentation in predefined language templates. Higher numbered macros (10 through 31) are not extension specific, but are otherwise similarly definable. They are reserved for the use of individual users.
<code>%open</code>	Open a new line following the current, similar to going to the end of the line and pressing Enter .
<code>%qPrompt\$</code>	Query for string to insert, using the string <code>Prompt</code> to prompt the user
<code>%repCNum</code>	Insert <i>Num</i> repetitions of character Ctrl .
<code>%restore</code>	Restored a saved position.
<code>%save</code>	Save a position for later restoration.
<code>%time</code>	Insert a formatted time string. (<code>hh:mm:ss</code>).

Table 6-2. Macros in Templates

<code>%toF</code>	Move cursor to the top (first line) of the file. This requires <code>%home</code> to assure positioning at the first character of the file.
<code>%xFuncCall\$</code>	Execute the Hawk API function call in <code>FuncCall</code> . This may be any function that could be assigned to a key or otherwise executed through <code>LibFunctionExec</code> .

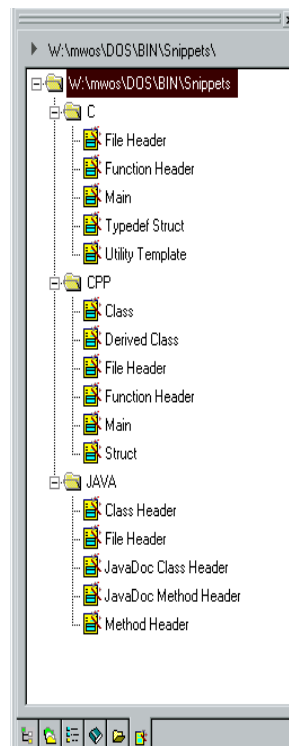
CodeFolio Snippets

CodeFolio Snippets are pre-defined chunks of code, specific to a programming language, that may be inserted into your current document. The style and structure of code or text that follows a Snippet remains unaffected.

Snippets have a default file extension of `.TPL`; configuration details for Snippets are stored in the file `CODESNIP.INI`.

To see a directory of the Snippets that are shipped with Hawk, click the **CodeFolio** tab located on bottom-right tab of the `Project` window. The directory tree is organized by language type, as shown in [Figure 6-1](#).

Figure 6-1. Code Folio Tab on the Project Window



In addition to using Snippets that are shipped with Hawk, you can add directories containing your own Snippets to the `CodeFolio` tab (refer to [Adding or Removing Snippets Directories](#) in this chapter). Snippets can also be edited to conform to your own needs (see [Editing a Snippet](#), also in this chapter).

Using an Existing Code Snippet

There are three ways to place an existing Snippet into a document. First, create a new document by clicking **File** -> **New** or opening an existing document by clicking **File** -> **Open** in the Client Area. Next, click the **CodeFolio** tab of the Project window (shown below).

Figure 6-2. Project Window Tabs



Execute a Snippet by completing one of the following:

- Left-click on the desired Snippet and drag it to your file. The Snippet will be placed where you drop the icon.
- Double-click on the desired Snippet. It will be automatically inserted at your cursor position in the file.
- Right-click on the desired Snippet. Select **Insert** from the pop-up menu. The Snippet will be inserted at your cursor position in the file.

Other examples of Snippets include the following:

TITLE.TPL

```
<HTML>
<HEAD>
<TITLE>Test</TITLE>
</HEAD>
<BODY>
<H1>Test</H1>

</BODY>
</HTML>
```

VBFUNCT.TPL

```
-----
''
' ** Test
'
' Parameters:
'
' Description:
'
' Returns:
'
-----
```


Adding a Code Snippet

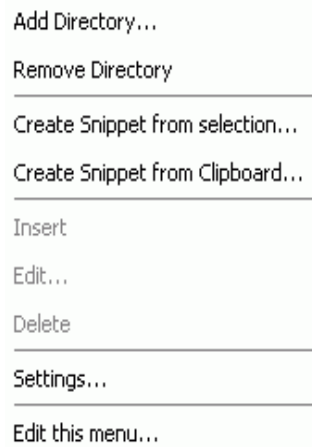
You may save portions of code that you have written as Snippets for later use. To do so, complete the following steps:

1. Create a new file.
2. Type or paste in the desired code.
3. Save the file with a `.TPL` extension in the `Hawk Snippets` directory.

Creating a Code Snippet from the Current Document or Clipboard

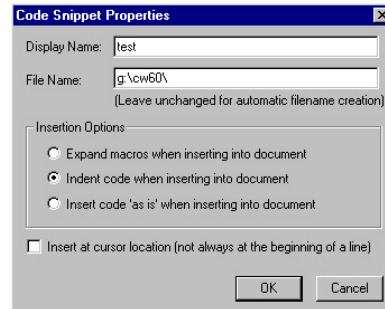
You can paste selected text from the current document, or text from the clipboard, and save that text as a new Snippet. To do so, complete the following steps:

1. Copy the desired text to the clipboard, or select the text in your document.
2. Right-click on the `Snippet` directory in which you want to save the new snippet, or drag the selection to the appropriate `Snippet` directory folder.
3. The following menu displays:



4. Select `Create Snippet from selection` or `Create Snippet from Clipboard`, as appropriate.
5. The `Code Snippet Properties` dialog displays (as shown in [Figure 6-3](#)). In this dialog define the display name and storage directory for your new snippet. You can also choose whether or not to expand macros upon insertion, whether or not to match the level of indentation at the insertion location, and whether or not to insert code "as is". If you want to insert the snippet at the cursor, mark the checkbox accordingly.

Figure 6-3. Code Snippet Properties Dialog



Deleting or Renaming a Snippet

You can delete or change the display name of a Snippet from the pop-up menu.

To permanently delete a Snippet from your system, complete the following steps:

1. Right-click on the name of the Snippet in the CodeFolio tab's tree directory.
2. Select **Delete**.

To reference the Snippet as a different name, edit the name inline.

Renaming a Snippet does not change the name of the underlying file; it changes the reference to the Snippet on the CodeFolio tab. To determine the actual name of the Snippet file, complete the following:

1. Right-click on the name of the Snippet in the tree directory.
2. Select **Properties**. The Code Snippet Properties dialog displays, with the actual filename in the File Name field.

Editing a Snippet

To edit the actual content of a Snippet, complete the following steps:

1. Right-click on the name of the Snippet in the CodeFolio tab's tree directory.
2. Select **Edit** from the pop-up menu. The Snippet is displayed in a separate window.
3. Edit as desired. Select **File** -> **Save** from the main menu to save your changes.
4. Close the Snippet file.

Adding or Removing Snippets Directories

You can add or remove a root level node from the tree directory on the CodeFolio tab. .



To create or remove a subdirectory of an existing Snippets directory, you must add/delete the folder outside of the CodeFolio tab.

To add a root level directory, complete the following steps:

1. Right-click on the directory under which the new directory should be placed.
2. Select **Add Directory** from the pop-up menu.

3. Browse for and select the desired Snippets directory on your system.

To remove a root level directory, complete the following steps:

1. Right-click on the directory you wish to remove from view.
2. Select **Remove Directory** from the pop-up menu.

When you remove a directory, you are not actually deleting the underlying files; you are simply removing the directory and files from this view. When you delete the last root-level directory, the system uses the default `snippets` directory.

Brace Matching and Brace Expansion

Brace matching support in Hawk works several ways, including the following:

- The current buffer is checked to see that braces are balanced. This form is useful for checking syntax.
- The text between a set of braces is highlighted. This form is useful in visualizing the scope of the block defined by the braces and indenting blocks.
- The matching brace is located and the cursor is moved to it, either momentarily or permanently.

All of the Brace Matching functions are available on buttons on Hawk's `Edit` toolbar. To turn on the `Edit` toolbar, mark the **Visible** option for the `Edit` toolbar on the **Customize** -> **Toolbars** -> **Toolbars** dialog.

These services are performed by functions assigned to keys or buttons. You may wish to customize these assignments or make new assignments. Key assignments may be made through the **Customize** -> **Keyboard** dialog. Button assignments may be made through the **Tools** -> **Customize** -> **Toolbars** dialog. More information on customizing buttons and keystrokes can be found in [Chapter 12 Customizing the Interface](#).

Brace Function: Finding Unmatched Braces

There is a function supplied with Hawk, called `Brace()` that processes the current buffer to look for unmatched braces. All braces matched within the file can be specified.

One way to invoke the `Brace` function is to use Hawk's `API Command Key`. The `API Command Key` accesses a prompt or dialog from which Hawk APIs can be interactively used to perform their associated function. To access the `API Command Key`, use one of the following methods:

- Press `(` if you are using the CUA keymap.
- Press `)` if you are using the BRIEF-compatible keymap.
- Select **API Command** from the **Tools** menu.

The `Command Key` is described in more detail in [Chapter 4 Command Key, Libraries, and General Environment](#)

To match all curly braces within a file, respond to the `Command:` prompt in the following manner:

Command: Brace TRUE font

Ignore braces in comments, omit the TRUE parameter, or supply FALSE as a parameter. The function shows its progress by displaying the number of the line it is processing on the status line. If the function finds an unmatched curly brace, the cursor is positioned on that curly brace.

Brace Highlighting

There are two functions that examine or operate on the text between matching curly braces and parentheses:

- The first function is `BraceMatchNext`; it looks for the next left brace or parenthesis, locates its mate, then highlights the text between them. On subsequent calls, this function finds pairs nested within the highlighted set, or pairs following the highlighted set.
- The second function is `BraceMatch`; this function operates similarly to `BraceMatchNext`, except that it searches for a match to the brace at the cursor position. If the curly brace at the cursor is a left brace or parenthesis, the function searches forward for its mate. If it is a right curly brace or parenthesis, the function instead searches backward for the mate. If it is neither a left nor a right parenthesis or if brace is at the cursor position, the function searches forward for the next matching set.



Since `BraceMatch` looks at the current cursor position, it is not effective for highlighting a series of blocks in sequence. On subsequent calls to this function, `BraceMatch` "finds" the pair that is already highlighted.

If the parameter to this function is omitted, it will look for braces only. Passing a non-zero parameter to either of these two functions will cause it to look for matching parentheses in addition to curly braces.

Brace Locating

The function `BraceFind` is provided for locating the brace, parenthesis, or square bracket that is the mate for the one at the cursor position. It does not create a highlight, but positions the cursor at the corresponding object.

The function also has what is termed a "kissing" mode; this can be activated by giving the function a TRUE or 1 value as a parameter. In kissing mode, the cursor moves to the corresponding object, but only momentarily. After a brief pause, the cursor returns to its original position. This provides a method for making sure that the appropriate block or clause is closed.

BraceFindEx()

`BraceFindEx()` is another brace-matching function that can be used from Hawk's API Command dialog (API Command on the Tools menu), or bound to a button or keystroke.

`BraceFindEx` is an extended brace search and match function. It has a number of flags that can be used together or individually for more flexible brace matching.

An example of how the `BraceFindEx` function follows:

Example

The following combination of flags works for finding braces within quotes:

```
BraceFindEx BRACE_BRACES -> BRACE_PARENS -> BRACE_FORWARD
```

Brace Expansion

Brace Expansion is a special key assignment for C and C++ language support. It is activated on the `Customize -> Language -> Tabs/Indenting` dialog. Brace Expansion works in the following way:

- Type a brace. It is automatically positioned. The closing brace is instantly inserted in the correct position and the cursor is placed between them.

`Brace Expansion` operates very much like the brace template in template expansion. The difference between template expansion and brace expansion is that brace expansion inserts braces whenever the left brace (`{`) key is typed, while template expansion does not insert braces until the space bar is pressed.

The sequence used by template expansion may be unnatural for some programmers; therefore, brace expansion provides an alternative. Brace expansion may also be useful for users who do not want full template expansion, but want the benefits of automatic brace insertion.

Align Beginning and End of Block

`Align <begin/end block>` is a brace-expansion feature in Hawk that automatically positions language constructs that begin and end blocks (such as braces in C and C++) as you type them.

Example

Set the alignment setting on the `Customize -> Language -> Tabs/Indenting` dialog to `Unindented Block`. The closing curly brace for the following text will be positioned directly beneath the open curly brace whether that brace is typed at column 3 or 30:

```
Void myFunc(void)
{
```

`Align <begin/end block>` is similar to `Brace Expansion`, except that it does not instantly insert both the closing and opening braces when a brace is typed. It simply realigns the closing brace as it is being typed. The checkboxes for `Align <begin/end block>` are located on the `Tabs/Indenting` tab of the `Customize -> Language` dialog and are only available if the language support for the associated file type supports it. The options for `Align <begin/end block>` and `Brace Expansion` are mutually exclusive.

Indenting

This section discusses spaces and tabs, indentation, and alignment.

Setting Spaces and Tabs

Before editing a file it may be necessary to control which characters are inserted when the tab key is pressed. In some cases, you may not want to have actual tab characters inserted in a file. Hawk has an option for setting the tab key to insert spaces rather than tabs on the `Customize -> Language -> Tabs/Indenting` dialog. To set the tab key to insert spaces, unmark `Use tabs`. Remember that all changes made in the `Language` dialog are language specific; therefore, be certain to highlight the file type of the language of choice before making any changes.

The `Tabs/Indenting` tab is also where you set the size of tab-stops. The string placed in the `Tab Columns` editbox describes where tab stops will occur. Tab stops are placed at the columns specified in the string. For evenly spaced tab stops, you need not specify more than two column numbers. Hawk calculates the interval between the last two column numbers and repeats that interval between tab stops all the way out to the number specified in the `Maximum Column` field. The column count begins with 1, not 0.

Example

A four-column tab stop would be set by inserting the following numbers: `5 9`.

It works in the following way:

The cursor starts in column 1. When the tab key is pressed, four columns are added, putting the cursor in the fifth column. When the tab key is pressed again, four columns are added, putting the cursor in the ninth column, and so on, thus explaining the reason for the "5" and "9".

Tab settings can also be set on a per-document or per-window basis in the `Document -> Manager -> Tabs/Indenting` dialog.

Seek Indentation and Smart Indenting

There are two Indenting features that can be enabled in the `Customize -> Language -> Tabs/Indenting` dialog that make editing tasks easier. The features are `Seek Indentation` and `Smart Indenting`.

- The `Seek Indentation` function searches backward for a line containing printable characters and duplicates its level of indentation. It has options for the type of white space to be inserted when the indentation occurs, whether the white space consists of spaces, tabs or virtual space.
- `Smart Indenting` automatically indents according to common usage for the line of the language being edited. If the word (or character) being typed is one for which the following line is commonly indented, the indentation occurs after pressing `Enter`.

Example

A line in a `.C` file might end with a curly brace, or begin with `while`. When you press **Enter** to terminate the line after the curly brace, the next line will automatically be indented.

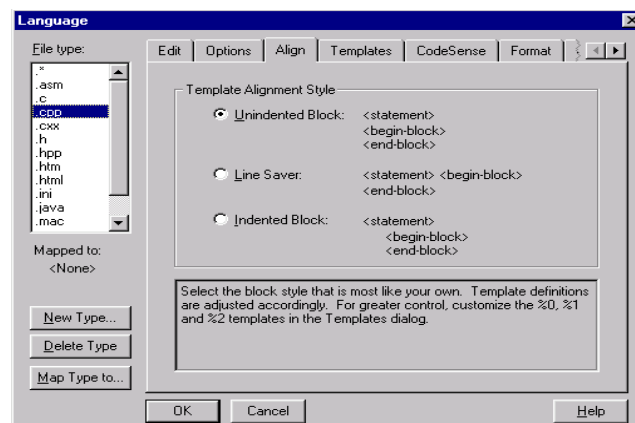
`Smart Indenting` may be thought of as an extension of the `Seek Indentation` feature. `Smart Indenting` may be used with `Seek Indentation` turned off, but leaving it on ensures the most consistent appearance.

`Smart Indenting` will not automatically indent lines that are being typed in a comment. The languages for which `Smart Indenting` is supported include C, C++, Pascal, dBASE, Visual Basic and Paradox.

Block Alignment

All the editing features that discussed thus far in this chapter deal with indenting blocks of code. However, preferences as to how indenting should occur may differ; thus, Hawk offers several different indentation styles to choose from. These styles are set on the `Align` tab of the `Language` dialog, as shown in [Figure 6-4](#).

Figure 6-4. The `Customize -> Language -> Tabs/Indenting` Tab



The `Tabs/Indenting` tab is used for selecting between several popular forms of block indentation to be used for brace expansion and template expansion. Choose from the following indentation styles:

- The `Unindented Alignment` places the block delimiters (`{` and `}`, `BEGIN` and `END`) at the same level of indent as the controlling statement that precedes the block. Statements between the block delimiters are indented.
- The `Line Saver Alignment` places the opening block delimiter (`{` or `BEGIN`, for example) on the same line as the controlling statement that precedes the block. The closing block delimiter (`}` or `END`, for example) is at the same level of indent as the controlling statement. Statements between the block delimiters are indented.
- The `Indented Alignment` is where the block delimiters (for example, `{` and `}`, `BEGIN` and `END`) are indented, compared to the controlling statement that precedes the block. Statements between the block delimiters are not indented.

Name Completion

Name Completion automatically complete strings that occur in the current document, the current project, or the symbols database as they are typed. (The Symbols database is used for storing information that is used for editing and navigating code and is described in [Chapter 10 Search and Replace and Other Navigational Tools](#)).

Name Completion is not turned on by default. To turn it on, mark **Name Completion** in the **Customize -> Language -> CodeSense** dialog. Mark related settings in the Name Completion group as desired.

The strings that name completion draws from are parsed as follows:

- Strings that occur in the current document are parsed by the **word delimiters** listed in the **Customize -> Language -> Options** dialog. The word delimiters are customizable.
- Strings that are contained in CodeSense databases are parsed by one of the following:
 - By file-type specific parsers that are listed in the **Outline Parsers** dialog. Click **Symbol Patterns** in **Customize -> Language -> CodeSense** to access the dialog.
 - By CodeSense (see the following topic [CodeSense](#) for more information).

To use Name Completion type the first few characters of the word to be completed and press **Ctrl-space**.

CodeSense

Hawk offers a feature called CodeSense that provides advanced "word" completion for .C, .CPP, .H, .HPP, .CS and .JAVA files. It works by gathering information about functions, methods, structures, unions, namespaces and other pieces of code in source files and then using the information for editing and informational purposes.

Information provided by CodeSense is automatically filtered based upon the file type of the file you are currently editing. (e.g. If you are editing a C document, CodeSense will not return Java information.)

The following topics describe CodeSense in detail and offer some troubleshooting tips in the event that the feature does not work as expected.

Where CodeSense Gets its Information

CodeSense gets its information from databases and from files that are open in Hawk. The next topics describe how that information is derived.

Library and Project Databases

The core of the CodeSense feature is composed of databases that store information gathered from parsed code. The information for the databases is found in the following places:

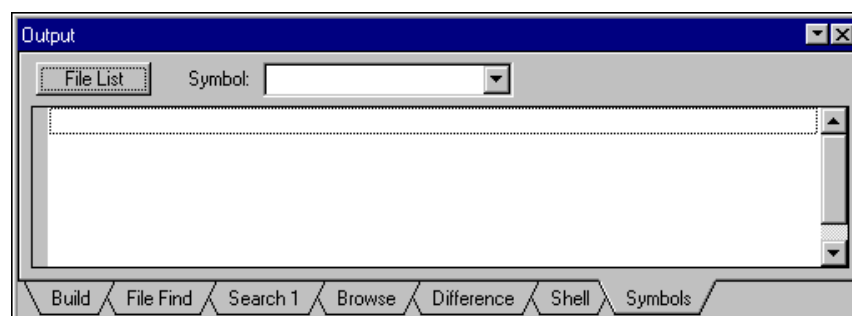
- From C/C++, C# and Java source files in any of the library databases listed in the `Customize -> CodeSense Global Configuration` dialog. These databases can reference directories of individual files or `.zip/.jar` files.
To display the properties of any scanned library database, highlight the database name and press the `Properties` button. If the `Properties` dialog indicates that you do not have an optimal version of the library database, especially if it is a full release different, you may want to press `Rescan` to rescan the database.
- From C/C++, C# and Java source files in Hawk projects. To display the properties of the current project's files that have been scanned for CodeSense use, plus files parsed by the outline parser, select `View Symbol DB Properties` from the status line CodeSense popup menu (right-click on the status line icon to access). If the `Properties` dialog indicates that you do not have an optimal version of the project database, you can right-click on again and select `Rescan symbol DB Files`. If the project database is a full release different, and therefore invalid, it will be rescanned automatically.
- From C/C++, C# and Java source files that are listed in the file list on Hawk's Output Symbols Window. Files in projects should already be in the list. The next topic describes how to add files to this list.

Add a File to Symbols Window File List

Some CodeSense information is gathered from files listed in the Symbols Window file list. Files that are in the current project will already be in the list. To add additional files to the list, do the following:

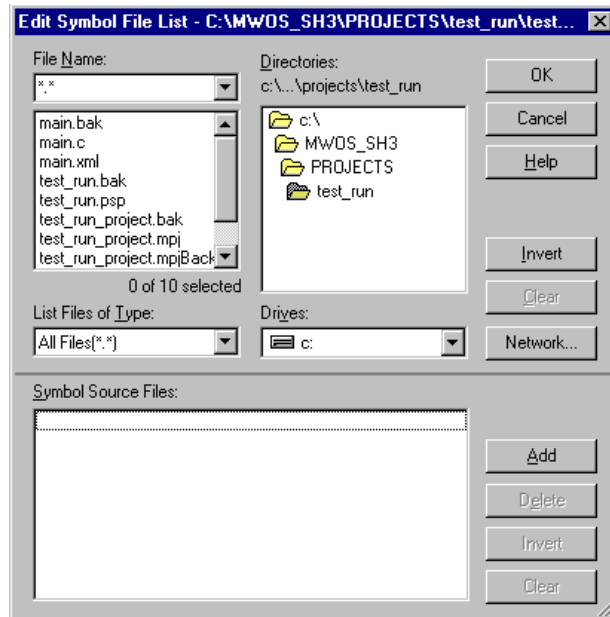
1. Click the `Symbols` tab of the `Output` window (located by default at the bottom of the Hawk screen).

Figure 6-5. Symbols Window



2. Click the `File List` button to access the `Edit Symbol File List` dialog and type in or browse for the file to be added to the list.

Figure 6-6. Edit Symbol File List



The following steps will also work to access the **Edit Symbol File List** dialog:

3. Right-click on the left most status bar icon to access the CodeSense popup menu.
4. Click **Edit Symbol DB File List** to access the Edit Symbol File List dialog.
5. Type in or browse for the file to be added to the list.

CodeSense for Files that are Open in Hawk

In addition to the information that CodeSense gets from library and project databases, it also gets information from C/C++, C# and Java files that are open in Hawk. The scanning process for open files relies on the option CodeSense DLL in the Customize -> Language -> CodeSense dialog. Make sure this option is selected for the appropriate file types (.C, .CPP, .CS and .JAVA).

CodeSense information that is gathered from open files is stored in memory rather than in databases. It is specific to the files that are open and will only be available while the files are open. This feature allows CodeSense to display information from files that have not yet been saved to disk.

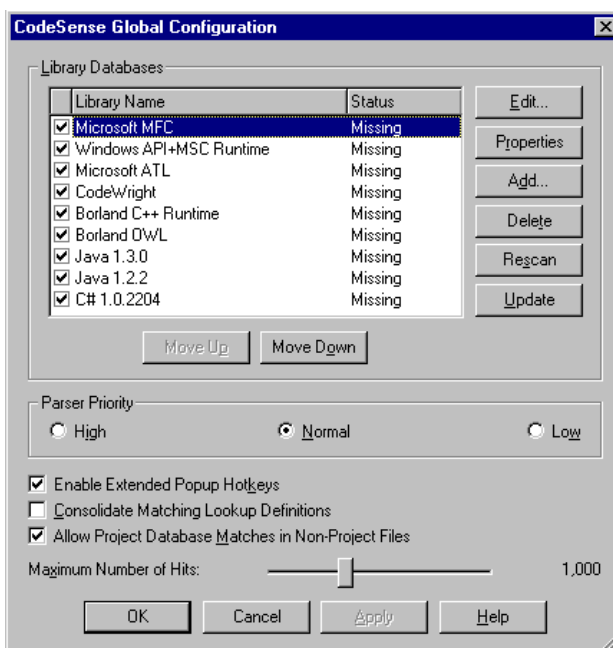
CodeSense Global Configuration Dialog

The bulk of the information for CodeSense is gathered from and stored in CodeSense library databases. CodeSense library databases are listed in the CodeSense Global Configuration dialog. They start out as directory locations for C/C++, C# and Java files. Several default library databases can be installed with the Hawk installation. They will appear in the dialog if they were installed. If the appropriate library database is not available with the Hawk installation, it can be created.

Regardless of the manner in which a library comes to be listed in the CodeSense Global Configuration dialog, it must always be scanned at least once in order for a database to be created. Libraries that are installed with Hawk have already been scanned. If a library has not been scanned, a "(missing)" message will appear to its immediate right. This means that it does not have an underlying database. Press **Rescan** to have CodeSense scan the selected library and create a database.

CodeSense will only perform lookups on scanned library databases that have been checked in the CodeSense Global Configuration dialog. Lookups for library databases are done in the order in which they are listed in the dialog. Moving libraries up and down in the list and unmarking those that are not immediately needed will speed up the lookup process.

Figure 6-7. CodeSense Global Configuration



The CodeSense Global Configuration dialog can be accessed from two alternate locations:

- From the **Customize** menu, by clicking **CodeSense Global Configuration**.
- From **Customize -> Language -> CodeSense**, by pressing **Global Configuration**.

Create CodeSense Library Database

In many cases, it will be necessary to create a new library database pointing to source code that CodeSense will scan. Do the following to create a CodeSense library database:

1. Click **Customize -> CodeSense Global Configuration**.
2. Click **Add** to bring up the **Add Library Database** dialog.

3. Give the library database a name, and then either type the path to the correct location of the source files or browse for the correct location (use the `Dir` button to browse for a directory, or the `File` button to browse for a .zip/.jar file). If you want CodeSense to include subdirectories at the location at which files will be scanned, put a check in the `Include Subfolders when Scanning` box.
4. Mark the option `Full Source` to have CodeSense scan all .C, .CPP, .H, and .HPP files in the specified CodeSense library database directories. Leave the option empty to have CodeSense scan only .H and .HPP files in the specified directories. This option does not apply to C# or Java. All .CS and .JAVA files will be scanned regardless.
5. Click `OK`, then `OK` again. CodeSense will begin scanning the files in the specified directory to create a database. Note that the newly-created library must be checked in the CodeSense Global Configuration dialog in order to make its database information available to Hawk.

Edit CodeSense Library Database Location

The default library databases provided with the Hawk installation may or may not point to valid directories, depending on a system's setup. Attempting to scan an invalid library database will result in a message prompting to change its directory. Do the following to change the location of a CodeSense library database:

1. Click `Customize` -> `CodeSense Global Configuration`.
2. Highlight the library to be changed.
3. Click `Edit`.
4. In the `Edit Library Database` dialog, either type the path to the correct location of the source files or browse for the correct location (use the `Dir` button to browse for a directory, or the `File` button to browse for a .zip/.jar file). If you want CodeSense to include subdirectories at the location at which files will be scanned, put a check in the `Include Subfolders when Scanning` box.
5. Mark the option `Full Source` to have CodeSense scan all .C, .CPP, .H, and .HPP files in the specified CodeSense library database directories. Leave the option empty to have CodeSense scan only .H and .HPP files in the specified directories. This option does not apply to C# or Java. All .CS and .JAVA files will be scanned regardless.
6. Click `Rescan`. CodeSense will begin scanning the files in the specified directory to create a database.
7. Click `OK`, then `OK` again. Note that the newly-edited library must be checked in the `CodeSense Global Configuration` dialog in order to make its database information available to Hawk.

Delete CodeSense Library Database

As mentioned, default library databases can be placed in the `CodeSense Global Configuration` dialog at installation. They are there for convenience only. They may not be useful for all systems. If inapplicable databases have been installed, it may be desirable to delete them. Do the following to delete a CodeSense library database:

1. Click **Customize** -> **CodeSense Global Configuration**.
2. Highlight the library to be deleted.
3. Press **Delete**. A prompt will ultimately appear asking if the underlying database should also be deleted. Click **Yes** to delete the database, then click **OK**...



Databases that were created from the files in a Hawk project must be deleted outside of Hawk, from Windows Explorer. Hawk should be closed before these databases are deleted. See the topic *Database Files* to find out what files to delete and where they are located.

Parser Priority/Resource Use

The CodeSense scanning process works on a background thread. It should not interfere with other Hawk processes. It has been known, however, to occasionally be time and resource intensive. If you find that your CPU usage is high because of CodeSense, you may want to set the parser priority to low. Mark **Low** in the **Parser Priority** section of **Customize** -> **CodeSense Global Configuration** to do this.

CodeSense Databases

Once the necessary CodeSense library databases have been established in the **CodeSense Global Configuration** dialog, and the **Rescan** button has been pressed if necessary, the libraries will be scanned for information that will be stored in databases for future use. Project and Symbols Window File List databases will be created automatically given that:

- **CodeSense DLL** is marked in **Customize** -> **Language** -> **CodeSense** (marked by default for **.C**, **.CPP**, **.H**, **.HPP**, **.CS** and **.JAVA** files).
- **Auto-update symbol database** and **Show Definitions in Symbols tab** are marked for the **Symbol file** option in the **Project Properties** dialog. This dialog is accessed by selecting **Project** -> **Settings** -> **Directories**
- A Hawk project containing **C/C++**, **C#** or **Java** files is open and/or **.C**, **.CPP**, **.H**, **.HPP**, **.CS** and **.JAVA** files are listed in the Symbols Window's file list.

Database Files

CodeSense databases are comprised of the following files:

```
FILE.CDX
FILE.DBF
FILE.FPT
SYMBOL.CDX
SYMBOL.DBF
SYMBOL.FPT
```

An additional file with a **.SBL** extension is also included among the database files, though it is kept separately from the other files. The files can be found at the following locations:

- Database files that are created for CodeSense library databases are located in the SENSEDBS\<<Library Name> directory of the Hawk home directory.
- The .SBL file is located in the Hawk home directory when no project is open and is called mwhawk.sbl.
- If a project is open, the .SBL file is located in the project's directory and is named <Project Name>.SBL.
- Database files that are created for Hawk projects are located in the <.SBL file path>\<Project Name.cs_> directory.

Database Corruption

Sometimes CodeSense databases can become corrupt. While CodeSense is designed to detect database corruption, it is not foolproof. If it detects corruption in a database created from a CodeSense library database, a "(Corrupted)" message will appear in the CodeSense Global Configuration dialog to the immediate right of the library.

If database corruption is suspected, all database files should be deleted for that database. CodeSense library databases can be deleted in one of two ways:

- By pressing the **Rescan** button for a selected library in the CodeSense Global Configuration dialog. This will cause the underlying database files to be deleted and the coinciding source files to be rescanned.

Use the following steps to rescan a CodeSense library database:

1. Go to **Customize -> CodeSense Global Configuration**.
 2. Highlight the library database to be rescanned.
 3. Press **Rescan**.
- By pressing the **Delete** button for a selected library in the CodeSense Global Configuration dialog. A message will ultimately appear asking if the underlying databases should also be deleted. Click **Yes** to delete the database. See the topic [Delete CodeSense Library Database](#) for numbered steps to delete a library database. Note that you will have to recreate the library database in order to rebuild it.

To rescan databases that are maintained for projects:

1. Right-click on the Symbols tab of Hawk's Output Window.
2. Select Rescan symbol DB files from the resulting popup menu.
3. To delete and rescan databases that are maintained for Hawk projects:

To delete and rescan databases that are maintained for Hawk projects:

1. Exit Hawk and go to Windows Explorer.
2. Navigate to the directory containing the project (.PJT) file.
3. Delete the project database. See the topic *Database Files* earlier in this chapter to get a list of the files and locations that make up the database. Project files will be rescanned automatically when the project is open so long as the following conditions are met:

- CodeSense DLL is marked in `Customize -> Language -> CodeSense`.
- `Auto-update symbol database` and `Show Definitions in Symbols tab` are marked for the `Symbol file` option in `Project Properties Directories` dialog.

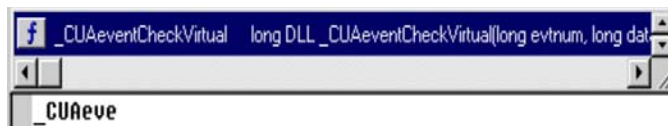
Using CodeSense

Once you have loaded the programming libraries needed to recognize your symbols and functions, you are ready to use CodeSense. Instructions for using the `Auto-list Members`, `Auto-type Info` and `Auto-parameter Info` features (in `Customize -> Language -> CodeSense`) are described in the following sections.

Name Completion

To have CodeSense complete or show a list of possible matches for symbols when `C-Space` is pressed, mark `Name Completion` in `Customize -> Language -> CodeSense`. Once marked, the option can be used by pressing `C-Space` at appropriate times while typing. The option is marked by default for `.C`, `.CPP`, `.H`, `.HPP`, `.CS` and `.JAVA` file types.

Figure 6-8. Example



When you highlight a definition in the list of possible matches, the Hawk status line displays the name of the database and file in which the definition can be found. If desired, click on the symbol's representative (Go To) image to open the identified file with the cursor positioned on the selected definition.

The following `Name Completion` options are available in the `CodeSense` dialog:

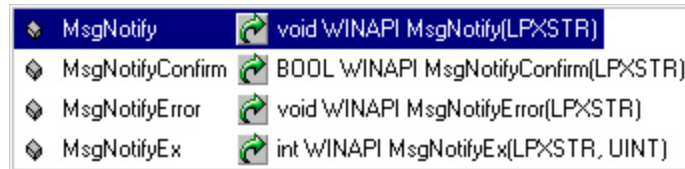
- `Complete Document`: Searches the entire document when looking for a string that matches the one being typed.
- `Symbol Database`: Searches through the `Symbols` database (`*.SBL`) for a matching string.
- `Cycle Through`: If multiple matching strings are found, matches are cycled until the desired string is found.
- `Show List`: If multiple matching strings are found, all matches and their definitions are shown in a drop-down list.
- `Even for Single Matches`: Matching string(s) are shown in a drop-down list, even if there is only one match, rather than automatically entering the string into your document.
- `Ignore Case`: Turns off case sensitivity when searching for strings that match what you have typed.

Auto-list Members (Standard)

The standard `Auto-list Members` feature helps you finish typing symbols. Once you have entered a few characters, complete the following steps:

1. Press **Ctrl-space** to bring up a list box of context-sensitive symbols to choose from (shown in [Figure 6-9](#)). The symbols begin with whatever you have typed so far; the definition of each symbol is provided in the right-hand column.

Figure 6-9. Sample Word Completion Drop-down List




2. Highlight the appropriate symbol from the list.
3. Press  (the Go button). The definition of the symbol within the library's source code is displayed (as shown below).


Figure 6-10. Sample Definition within Library Source Code

```
void WINAPI MsgMessage( LPXSTR );
void WINAPI MsgNotify( LPXSTR );
BOOL WINAPI MsgNotifyConfirm( LPXSTR );
```

4. Press **Enter**. The highlighted symbol is entered into your document automatically, completing the word you began.

Auto-list Members (Special Types)

CodeSense handles certain types of symbols (structs, classes, unions and enums) a bit differently. To use the `Auto-list Members` feature for these symbols, complete the following steps:

1. Type in the symbol, followed by a ":", "->" or "::" as appropriate. The latter triggers a drop-down list (similar to the one shown above) with all the members of that symbol and their definitions.
2. When you see the appropriate symbol, highlight it and press  (the Go button). The definition of the symbol within the library's source code is displayed.
3. Press **Enter**. The highlighted symbol is entered into your document automatically.

Auto-type Info

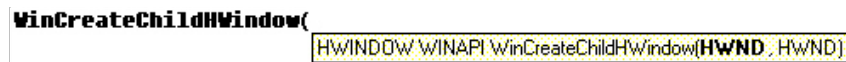
To use the Auto-type info feature, simply hover the mouse over the desired symbol to display a tooltip defining that symbol.

Auto-Parameter Info

To help you accurately enter function parameters, CodeSense displays the complete function prototype in a tooltip. As soon as you enter a left parenthesis to start writing a

function call, CodeSense offers the function prototype with the current parameter highlighted in bold. A sample is provided below.

Figure 6-11. Sample Function Call Tooltip

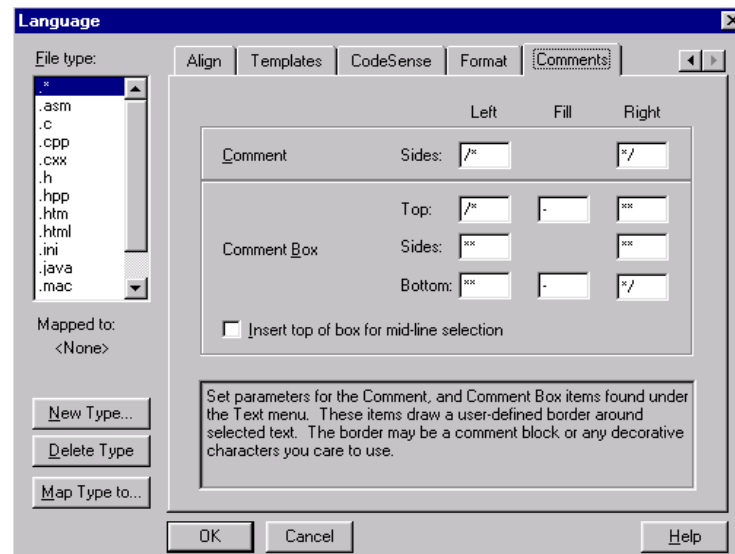


Once you type a right parenthesis or move your cursor away from the function call, the tooltip disappears.

Comments and Comment Boxes

There are two items on the **Text** menu that are used for automatically inserting comments into your code; these are the **Comment** and **Comment Box**. These menu items draw a user-defined border around selected text. This feature makes it convenient to write the text of a comment employing word wrap, then to add the comment characters later. The border may be a comment block or any decorative characters you care to use. The strings used for these borders are defined on the **Comment** tab of the **Language** dialog, shown in [Figure 6-12](#).

Figure 6-12. Comments tab of the Customize -> Language -> Comments Dialog



Define the look of **Comments** and **Comment Boxes** as follows:

- The **Comment** item adds a purely functional left and right border, which may serve as the beginning and end of a comment. The default for this item is to add `/*` to the beginning of each line or line segment in the selection, and to add `*/` at the end of each.
- The **Comment Box** item does the same, but adds a decorative top and bottom line. This type of comment is usually used for an eye-catching multi-line comment at the beginning of a function or file. It is primarily for line selections, but may also be used for column or stream selections.

The left side of the border is placed at the beginning of the line, or, in the case of a column selection, in the first column of the selection. The right side of the border is placed at the first tab stop to the right of the longest line in the selection.

HTML Editing

Hawk provides several features that make it ideal for HTML editing. These features are described in the following sections.

HTML Language Support

When HTML files are opened in Hawk, ChromaCoding is immediately available. All HTML keywords, comments, and strings are colored instantly. Template expansion (see [Templates and Brace Expansion](#)) is also available for automatic insertion of opening and closing tags with correct cursor positioning to allow for quick completion of statements.

HTML Popup Menu

An HTML popup menu is available whenever you press the shift key and right-click on an open HTML document. The menu contains a number of commonly used HTML tags that can be automatically inserted into the file being edited. If you click on the `New Document` item, a basic template for a new Web page will be inserted. You can assign the popup menu to another key or mouse combination by using the `Customize -> Keyboard` menu and assigning the command `'DlgMenuPopup HTML.MNU'` to your favorite keystroke. Doing so adds an entry to the `KmapAssign` section of Hawk's configuration file, `mwhawk.ini`.

The following example can be found in the `KMapAssign` section of `CWRIGHT.INI`:

```
[KMapAssign]
KmapAssign="<Shift-mouse_right_click>", "DlgMenuPopup HTML.mnu"
```

Items on the popup menu can be changed by using the `Edit this` menu item on the popup menu, or by editing the `HTML.MNU` file directly (See [Chapter 12 Customizing the Interface](#)). Edit `HTML.TPL` to change the way the `New Document` item on the popup menu behaves.

HTML Viewing – HTML WYSIWYG Editor/Viewer and Web Browser Interface

There are two methods for displaying Internet files being edited in Hawk. They include the following:

- An HTML WYSIWYG Editor/Viewer
- An interface that displays files being edited in certain installed web browser(s)

These viewing methods are implemented through the installation of two Add-Ons that can be loaded in Hawk's `Customize -> Libraries` dialog. The process for installing the Add-Ons is described next.

Viewing Internet Files: Installation Instructions

To install Hawk's HTML viewer or the web browser interface, do the following:

- Go to **Customize** -> **Libraries** and mark one or both of the following boxes:
 - **HTML WYSIWYG Editor/Viewer**
This installs CWHTML.DLL, to enable Hawk's WYSIWYG HTML editor
 - **Web Browser as Viewer**

This installs CWWEB.DLL, which turns on Hawk's web browser interface. During startup, this option searches for installed browsers and configures itself to interact with any that are found. If no installed browser is found or supported, the DLL may be configured manually. See the topic **Web Browser Interface** in Hawk's online help for information on configuring and using the web browser interface.

- If neither of the options described above are listed in the **Libraries** dialog, click the **Add** button and select the appropriate DLL(s) from those listed in the Hawk directory. Both installation methods cause Hawk to load the library into memory. It also adds lines to the `mwhawk.ini` file so that the DLL(s) will automatically load when Hawk starts up. Examples of the entries in `mwhawk.ini` follow:





```
[LibPreLoad]
LibPreLoad=cwwweb.dll
LibPreLoad=cwhtml.dll
```

HTML Toolbar

Hawk has an HTML toolbar which can be turned on in the **Customize** -> **Toolbars** dialog. It links to the API Assistant and available HTML keywords (see [API Assistant](#)).

Using HTML WYSIWYG

Once you have loaded `HTML WYSIWYG editor/viewer`, do the following to use it:

1. Load an HTML file.
2. Make the `HTML` toolbar visible (if it is not already) in **Customize** -> **Toolbars**.
3. If not already depressed, depress the **Toggle viewer**  button on the `HTML` Toolbar. This turns on the WYSIWYG view. Once the button has been depressed, several other buttons are made available. The buttons provide the following functions:
 -  View and edit the current file in a split code/WYSIWYG window.
 -  View the current file in a full WYSIWYG window.
 -  View the current file in a full code window.


- The remaining buttons perform various HTML WYSIWYG and code-editing tasks such as inserting tags, setting colors, and inserting tables and graphics. They will be alternately available and unavailable as they apply to the view (code or WYSIWYG) that currently has focus.
- You can view/test your document in Internet Explorer, Netscape Navigator, Mosaic or Opera, if those browsers are loaded on your system. Add the corresponding buttons to your HTML toolbar from `Customize -> Toolbars -> Select Buttons`. If your filename or path contains a space, and you are using Opera or Mosaic, open the file directly from the browser instead using the `File -> Open` command.


Editing Capabilities

HTML WYSIWYG editing provides the following capabilities:

- Edit HTML files as normal documents. Text cut or copied from the WYSIWYG window is available on the `Clipboard` tab of the `Output` window.
- Quickly insert HTML tables and graphics.
- Resize HTML tables and graphics with mouse drag operations.

Differencing/Merging

By default, changes made in the code window must be saved or undone before editing in the WYSIWYG window. Conversely, changes made in the WYSIWYG window must be updated or abandoned before the code window can be edited. The `Update` button  transfers changes one-way, from the current window to the opposite view.

If you wish to enable differencing/merging (so the `Update` button combines changes from both views and updates both views), press the `HTML Options` button  to invoke the `HTML Configuration Options` dialog and mark `Enable Differencing/Merging`. Also on this dialog:

- Mark `Enable WYSIWYG Editing` to enable editing in the WYSIWYG window.
- Mark `Open Viewer in Edit Mode` to open the WYSIWYG viewer in editing mode by default.



You can also set these flags programatically using the `HTMLConfigFlags` API function, described under `Help->API Assistance On`.

Differencing/merging in the HTML WYSIWYG window will be disabled if any of the following load errors are encountered:

- Document contains one or more frame sets.
- Document contains server script too complex to be parsed.
- Document contains multiple `<BODY>`, `<HEAD>`, `<TITLE>`, or `<HTML>` tags.
- Document contains server script inside a `<LISTING>` tag.
- Document contains server script inside a `<LABEL>` tag.

- Document contains server script inside a `<TEXTAREA>` tag.
- Document contains server script inside a `<SELECT>` tag.

Hex Editing

A frequently used Hawk editing feature is Hex-mode editing. Hex mode allows you to view and edit a file in hexadecimal values. You may edit the file by entering values for each byte, or by typing the ASCII equivalent. Do one of the following actions to enter Hex mode.

- Press `CSH` in the `CUA` or the `BRIEF` keymap.
- Press `$` in the `vi` keymap.
- Use the button on the `EDIT` toolbar (described briefly in [Chapter 12 Customizing the Interface](#)) that switches the current document to Hex mode.

When you enter the Hex mode, the `T` key is redefined. It may be used to switch from the Hex values to their ASCII equivalent, and back again. To return to Standard mode from Hex mode, press the `X` key.

Hex mode displays the document offset at the left, 16 bytes of hex values in the middle, and their ASCII equivalent to the right. The 16 hex values are divided into four groups of four. Each value has two digits: a high nibble and a low nibble.

Insert vs. Overtyping Mode

Most Hex mode editors cannot change the length of a file, and therefore are always in Overtyping mode. Hawk can insert characters while in Hex mode, and it will be in Insert mode if you were in Insert mode before entering Hex mode.



If the file or portion of a file you are editing is length-sensitive, you will probably want to toggle to overtype mode.

When entering text in the ASCII portion of the Hex mode display, insertions appear as they do when you are in normal text mode, except that the text is in a rather narrow column. When in the Binary portion of the Hex display, each byte is displayed as two hexadecimal numbers, the high nibble on the left and the low nibble on the right. When you type numbers (0 to F) in these positions, it changes or inserts the character whose numerical equivalent you have typed.

You cannot insert a nibble; you must insert an entire byte. Therefore, when you type at the high nibble position, both nibbles of the byte are inserted. The low nibble is initially zero. The cursor is then positioned over the low nibble. The next number you type replaces the value of the low nibble. Thus, as you type you will alternately see two nibbles inserted and one nibble overtyped.

Handy Hawk Hex-Editing Features

Keep in mind the following Hawk features when using Hex mode:

- Highlighting text in Hex mode will highlight the corresponding ASCII side.
- Hex characters can be copied, cut, and pasted in the same manner that ASCII characters can, but the document to which the hex characters are being pasted must be in hex mode in order for the pasted text to appear in hex; otherwise, hex characters are pasted in ASCII.
- When the cursor is in the hex side of the document, a "black on yellow" color mark will appear for the character opposite the cursor in the corresponding ASCII side. The color mark will appear in the same way in the hex side of the document if the cursor is in the ASCII side. This helps track the hexadecimal equivalent if in ASCII mode or the ASCII equivalent if in hex mode.
- The cursor will wrap on either editing side to the next/previous line when the right/left arrow key is used..



Tips for searching and replacing binary and hex characters can be found in [Chapter 10 Search and Replace and Other Navigational Tools](#).

Clipboard and Scrap Buffers

In Hawk, copy and paste operations can be stored in the Windows Clipboard or in Hawk's Scrap Buffers. To toggle between Clipboards and Scrap Buffers, select one or the other on the Edit pull-down menu. In both cases, multiple buffers are allowed.

Scrap Buffers

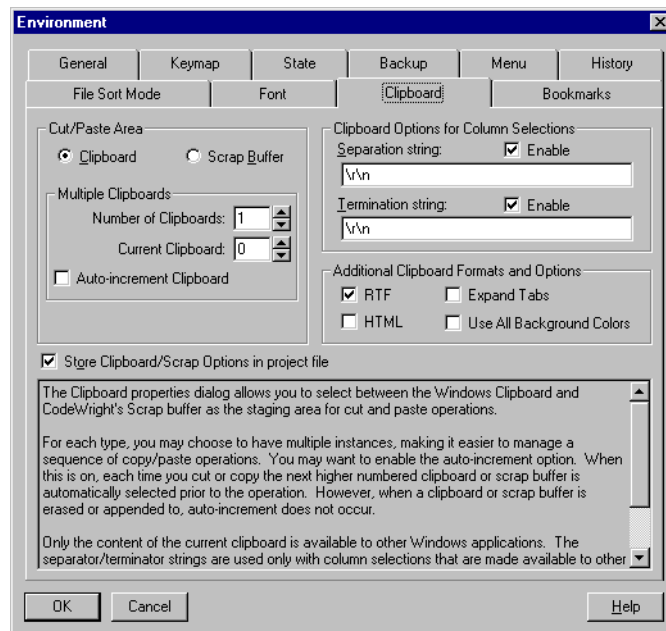
Hawk offers the option of using Scrap Buffers as the staging area for local cut and paste operations. Scrap Buffers can be used within or between edit documents in Hawk much as the Windows Clipboard is used for cut and paste operations between applications.

Hawk also supports the Windows Clipboard for transferring data between programs. Choose the repository you want to use from the Edit menu.

Multiple Clipboard/Scrap Buffers

Hawk allows the use of multiple Clipboards or Scrap Buffers. The number of buffers available can be defined by clicking **Customize** -> **Environment** -> **Clipboard** dialog (shown in [Figure 6-13](#)). The default is one.

Figure 6-13. Tools -> Customize -> Environment -> Clipboard Dialog



The Clipboard tab also has an Auto-increment Clipboard/Scrap Buffer option. This option works for both the clipboard and the scrap buffer. The clipboard/scrap buffer to use is incremented immediately prior to copying or cutting new content. If the highest numbered clipboard/scrap buffer is already the current one, the zeroth one is used next.

Check the **Store Clipboard/Scrap Options in project file** option to store these Clipboard/Scrap settings with the project currently open. If the box is checked but no project is open, options selected on the Clipboard tab are stored as default settings for use with future projects.

Select **RTF** to save information to the clipboard in Rich Text Format, RTF, as well as standard text format. Select **HTML**, Hypertext Markup Language, to save information to the clipboard in HTML format in addition to standard text format. These options are only available for clipboard storage.

Select **Expand Tabs** to replace tabs with the proper number of spaces to reach the next tab stop. This option works when copying content from Hawk, and uses the tab stops in the current Hawk document at the time the clipboard content is rendered. Rendering occurs when the first application requests the clipboard content, such as when the information is pasted into another application or when Hawk exits.

The Expand Tabs option will work as long as it is selected prior to the clipboard content being rendered. Currently, it only affects RTF clipboard content; tabs are always expanded for HTML content.



This option is only available for clipboard storage.

If the Windows® Clipboard Viewer application is running, it may render the clipboard contents immediately, rather than at the time of the paste operation.

When RTF or HTML format is requested, `Use All Background Colors` may be selected to preserve the background color settings from the View Setup in the clipboard content copied from a Hawk document.

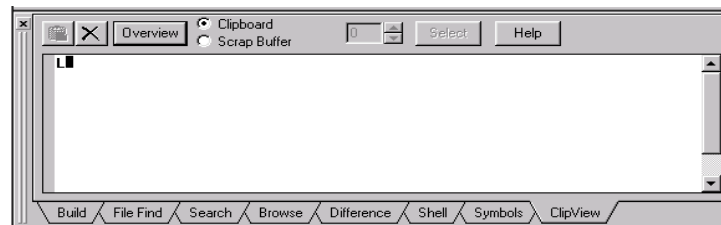
If `Use All Background Colors` is not selected:

- The clipboard content uses the default background color of the external application into which you are pasting; and
- A different background color is only generated in the pasted content if the background color of a particular character in Hawk is different from the background color for the Text element of the controlling View Setup.

Clipboard/Scrap Viewer

An extra tab for viewing the contents of the current `Clipboard` or `Scrap Buffer` can be turned on for the `Output` window. The tab is called `ClipView`; it is displayed in [Figure 6-14](#). Make the `ClipView` tab of Hawk's `Output` window visible by marking the `Clipboard/Scrap Viewer` option in the `Tools -> Customize -> Libraries` dialog.

Figure 6-14. The ClipView Tab



The `ClipView` tab provides the following functionality:

- The `Select` button selects the currently displayed clip or scrap buffer to be the default repository for copy/cut operations. The button is not available if the buffer displayed in the `ClipView` window is already the default repository.
- The `Paste` button pastes the contents of the `ClipView` window.
- The `Delete` button deletes the contents of the `ClipView` window.
- The `Overview` button allows you to view all `Clipboard/Scrap Buffers` available (you may have to use scroll bars to see additional buffers). When you are in `Overview` mode, the `Detail` button returns you to the current clipboard/buffer view only.
- The `Clipboard` and `Scrap Buffer` radio buttons allow you to view the contents of `Clipboards` or `Scrap Buffers`, and to select either as the current repository for copy/cut operations.
- The `Increment Clipboard/Buffer` change the number in this field to view the contents of another `Clipboard` or `Scrap Buffer`.

API Assistant

The `API Assistant` is an editing tool for Hawk that completes and inserts functions, with their appropriate parameters, at the cursor position in the current document.

Using the API Assistant

1. Begin using the API Assistant by selecting `API Assistance On` from the `Help` menu. If there is a word near the cursor at the time that the `API Assistant` is accessed, Hawk will attempt to locate assistance for that word. If no word is nearby, the `Find Assistance` dialog box appears.
2. Enter either the name or a part of the name of the desired function. If more than one occurrence of the word is found, the `Find Assistance` dialog box displays a selection to choose from. Double-click on the desired function name or select it and click `OK`.
3. A dialog box for the function appears. Enter the parameters for the function in this dialog box. There is an edit box for each parameter used by the subroutine or function. Initially, each edit box contains the parameter type for each parameter. Also, some functions will have an example parameter name for that parameter displayed with the parameter type. This makes it simple to ensure that all necessary parameters are supplied, and are of the correct type. If additional information is needed, press the `Help` button to see the `Windows API Help` entry (or other help file entry) for that subroutine or function.

When the form is completed, press `OK` and the information provided is assembled into a complete function call, including commas and parentheses, and inserted into the document.

Using the Checkboxes

If there are check boxes in the function or subroutine's dialog box, they are usually associated with a numeric parameter, such as an integer. The values checked, as represented by the predefined labels, are `ORed` together in the resulting function call.

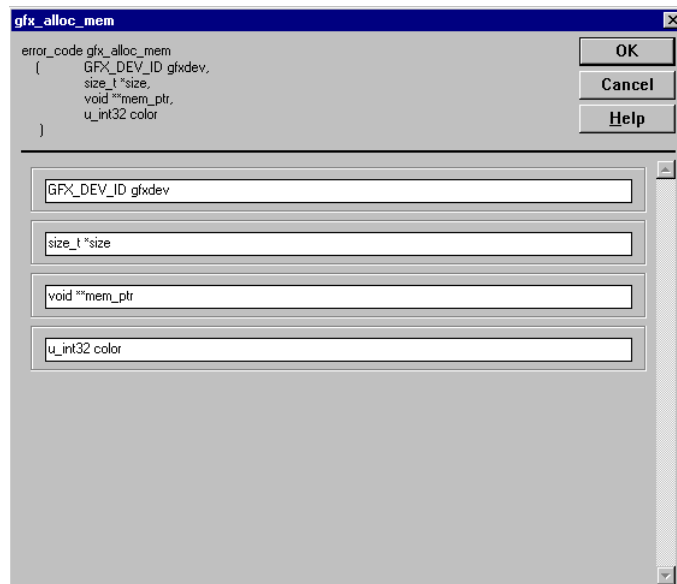
API Assistant Example

Refer to the following example and [Figure 6-15](#) for information on using the `API Assistant`:

The `API Assistant` can be used to help complete the `gfx_alloc_mem` function from the OS-9 MAUI® API:

1. **Access `API Assistance On` from the `Help` menu to bring up the `Find Assistance` dialog.**
 1. Type `gfx_alloc_mem` in the `Keyword` dialog box and press `Enter` to bring up a form containing the `gfx_alloc_mem` function and empty edit boxes to fill in for the function's parameters. Check boxes for the function's numerous flags are also available.

Figure 6-15. The API Assistant (gfx_alloc_mem()) Dialog



2. Fill in the necessary parameters.
3. Once the desired parameters have been inserted, click **OK** to insert the `gfx_alloc_mem` function into the current file.

Currently, API Assistance is provided for the OS-9 API, Standard C Library functions, Sun Java, HTML, and the Hawk API.

Modifying the Database

One of the sample “tools” placed at the end of the `TOOLS` menu is the `API Database Editor`. The `API Database Editor` is for modifying and making minor additions to any of the API Assistant’s databases. The database editor provides the primary method by which parameters may be presented as a series of choices, rather than simple edit boxes. Presenting the parameter as a series of choices assists the user in remembering or selecting a reasonable value for the parameter, and it tends to validate the choice.

API Databases

The databases supplied as of this writing are described in [Table 6-3](#).

Table 6-3. API Databases

Filename	Description
OS9_*.TDB, SOFTSTAX.TDB, OS-CALLS.TDB MAUI.TDB	Contains information about OS-9 API function calls.
HAWKAPI.TDB	Contains information about Hawk API function calls.

Table 6-3. API Databases

C . TDB	Contains information about Standard C library functions.
RWCLASS . TDB	Contains information about the C++ class library
JAVA . TDB	Contains information about Java functions.
SUNJAVA . TDB	Contains information about Sun Java functions.
HTML . TDB	Contains information HTML tags.

Using Help in Hawk

Hawk's help system can be configured to support multiple help files for easy access to help topics from different environments. When Hawk's help has been properly configured, pressing **!** when the cursor is positioned on a word in the current file will show a list of help files in which that particular word was found. The appropriate topic can then be chosen from the various files presented.

Indexing Help Files

Hawk ships with its own help files (CWHELP . HLP, CWAPI . HLP, CWAPPBAS . HLP, and CW_FAQ . HLP). You can also index help files from other software packages so that you can access that help from Hawk.

Files added to this index must end with one of the following extensions:

- . HLP
- . MVB (Microsoft C 4.0)
- . IVT (Microsoft Visual C 5.0)
- . HTML (Microsoft Visual C++ 6.0)

To index your . HLP help files, open the `Configure Help Index` dialog by selecting `Help -> Configure Index File`, select `Add` in the `Configure Help Index` dialog, browse for the help file you need and click `Open`. Then choose `Close` from the `Configure Help Index` dialog. Hawk will add this help file to the help index (MWHAWK . IDX).

Once the index has been created, pressing **F1** on a word that is contained in the newly-indexed help file will cause the word to appear as a choice in a help window. If no help topic is found, a dialog comes up listing all of the available help files. Any of the help files listed can be selected and a second search can be initiated within that help file.

Hawk can also be set up to display help from files other than . HLP files (such as . MVB, . IVT, and . HTML help files). These processes will be discussed next.

Setting Up Hawk to Integrate with MSVC 6.0 Help Files

With the release of Visual Studio 6.0, Microsoft changed the format of its online help and documentation system. The new system is based on Microsoft HTML Help. Hawk provides the ability to access MSVC's context sensitive HTML help using the **!** key.

To set Hawk up to access HTML help files, go to **Help** -> **Configure Help Index**.

1. Click the **HTML Help Setup** button.
2. Check **Use HTML Help Viewer as default when keyword not found**.
3. Click the **OK** button.
4. Click the **Save** button.

When configured, the **Search For Help On** item in the **Help** menu (or **!** key) will be connected to Visual Studio 6.0 Help. The first time you invoke help, the **HTML Help Viewer** is launched, and a keyword lookup is performed on the topic of interest. Subsequent help invocations bring the **HTML Help Viewer** to the foreground, and perform the lookup.

Setting Up Hawk to Integrate with MSVC 4.0 (.MVB) Help Files

To set Hawk up to use MSVC 4.0 help files, complete the following:

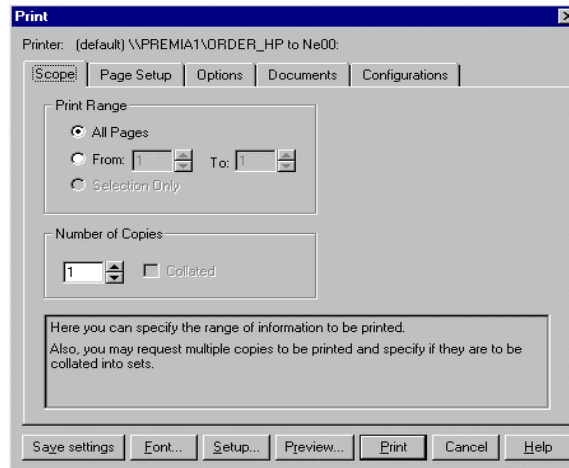
1. Go to **Help** -> **Configure Index File**.
2. Click **MVB Setup**.
3. In this dialog complete the following:
 - Uncheck **Use As Default when keyword not found**.
 - Uncheck **Use DDE**.
 - Remove the filename for the default **.MVB** file from the **Topic** field.
 - Enter the following macro in the **Topic** field: **\$(MVBFILE)**.
4. Return to the **Help** -> **Configure Index File** dialog.
5. Press the **Add** button. You will see that Hawk now allows you to index multiple **.HLP** and **.MVB** files. Add any necessary **.MVB** files.

Once the **.MVB** help integration setup is complete, position the cursor on a keyword and press **!**, or enter the keyword in the **Search for Help On** dialog, to access all the help (**.HLP** and **.MVB**) files listed in the index. A dialog box showing the help files found will be presented from which to choose the help file to be viewed.

Printing

Hawk's print dialog has options that make printing more flexible and efficient. Documents can be printed in color (if a color printer is available), line numbers can be added, long lines can be wrapped or not wrapped (whatever the choice may be), and sets of configurations can be created and stored with individual print configuration names. A quick description of some of the features available in Hawk's print dialog is shown below.

Figure 6-16. Print Dialog



Print Configurations

Hawk has the ability to save named print configurations. The `Configuration` tab of the `Print` dialog has a listbox that contains existing configuration names. Create a configuration by choosing some options to be stored, typing a new configuration name in the `New Configuration Name` box, then clicking `Create`.

Selected `Print Configurations` can be deleted, updated, or loaded. Both the `Load` and `Update` operations modify the 'current configuration'.

Paper Selection Override



On the `Page Setup` tab of the `Print` dialog, there are three checkboxes available that allow you to override the default paper size, source tray and orientation settings for your printer. If not supplied, the default settings will be used.

Color Printing

The `Options` tab of Hawk's `Print` dialog has options for printing in color. The colors printed will be those that are seen on the screen. It is also possible to distinguish colors using font styles (bold, italics, underlined). Font styles and colors can be used separately or together when printing.

Do not confuse font-style printing with the printed font. Documents printed in Hawk will always display the fonts that are set up in the `Print` dialog, not the fonts that are displayed on the screen. The font styles are only for bolding, italicizing, or underlining text being printed. Font styles can be specified in the `Font Style` group of the `Customize -> View Setups -> Font` dialog.

Print Preview

Hawk has a `Preview` button in its `Print` dialog that allows you to see what the document will look like before printing. The `Print Preview` has buttons for zooming  and for moving between pages in the document .

Multi-Copy Printing

Hawk's multi-copy printing feature can use the multi-copy capabilities of a printer, if present, in non-collated mode. The configuration options are on the `Scope` tab of the `Print` dialog.

Printing Line Numbers

The numbering option on the `Options` tab of the `Print` dialog has a line-numbering checkbox. Unmark the check box to turn off the number increment.

Wrapping Long Lines in Printed Documents

Two checkboxes pertain to line wrapping on the `Options` tab of the `Print` dialog: `Wrap Long Lines` and `Mark Wrapped Lines` (dependent on `Wrap Long Lines`). The first enables wrapping while the second controls the printing of a wrapping indicator on each line continuation.

Wrapping Long Lines in Printed Documents

Two checkboxes pertain to line wrapping. The first enables wrapping while the second controls the printing of a wrapping indicator on each line continuation.

Print Headers and Footers

The header and footer groups on the `File -> Print -> Page Setup` dialog permit header and footer text to be defined for each printed page.

Next to the `Header` and `Footer` edit boxes are buttons with right-pointing triangles. Press a button to display a menu showing macros available for headers or footers, respectively. Selected macros are inserted at the cursor position within the header or footer edit box automatically.

Available print macros refer, in a generalized way, to the page number (`%p`), current date (`%d`), current time (`%t`), and name of the file you are printing (`%f`). Place these keywords at the location within the header or footer string where you want the corresponding information to appear.

Formatting Headers and Footers

You may define up to three fields within each header and footer: left-justified, centered and right-justified. These fields are defined by the placement of semicolons. The first portion or field is left-justified. The field following the first semicolon is centered, and the portion following the second semicolon is right-justified. See the format diagram below:

```
<left-justified portion>;<centered portion>;<right-justified portion>
```

Any field in this format may be empty.

Examples

If you wanted to left-justify the filename but right-justify the date, your format would look like the following:

```
%f ; ; %d
```

Based on a filename `C:\CW32\F00.C` printed on `1\1\1999`, the printed results of the above header/footer macros would look like the following:

```
C:\CW32\F00.C 1\1\1999
```

To center a page number, your format would look like the following:

```
 ; %P
```

The printed results of the page number for page 1 would look like the following:

```
 1
```

Macros to be used in headers and footers are listed in [Table 6-4](#).

Table 6-4. Macros in Headers and Footers

Macro	Expansion Value
%d or %D	date
%t or %T	time
%f or %F	filename
%v or %V	volume
%r or %R	root
%e or %E	extension
%b or %B	basename
%p	page n of m
%P	page number
%x or %X	project path
%y or %Y	project root

Side-by-Side Difference Printing

A Print button is available from the Side-by-Side Difference Window (on the Difference tab of the Output Window) when the following conditions are true:

- The window is in View Mode.
- Both sides of the Window are in the same mode (i.e. normal, compressed), and neither is in hex mode.

Modified Print Dialog

When you press the Print button from the Side-by-Side Difference Window, a modified Print dialog entitled Print - Side by Side Difference displays. This dialog operates much the same as the standard Hawk Print dialog, with the following differences:

- The Documents tab is disabled.
- The Selection Only radio button on the Scope tab is disabled.
- The Print Two Up, Wrap Long Lines and Mark Wrapped Lines checkboxes on the Options tab are disabled.

Functionality

Printing the contents of the Side-by-Side Difference Window is similar to printing a regular document in two-up mode, except that the left side will contain the Reference document/file, while the right side will contain the Target document/file. Differences between the documents are shown as follows:

- Lines that are "deleted" in one side will display as a minus sign only.
- Lines that are "added" on one side will print, preceded by a plus sign.
- Lines that are modified will be preceded by an asterisk.

7

Projects, Project Spaces, and Workspaces

This chapter describes how to use the Hawk project, project space, and workspace facility. The following sections are included:

[Concepts of the Project Menu](#)

[Creating a Project Space](#)

[The Project Properties List Box and Project Settings](#)

[Creating and Modifying Projects](#)

[Project Setup Checklist](#)

[Using Projects and Project Spaces](#)

[Configuration and State Hierarchy](#)

Concepts of the Project Menu

The following section briefly defines the items included in the `Project` menu.

Project Space

Project spaces store sets of projects. Before a project can be made, a project space must be created. Project spaces allow multiple projects to be displayed at one time. The `New`, `Open`, and `Close` items on the `Project -> Project Space` submenu are for creating, opening and closing project spaces. The `Project Space` submenu also contains the `Add New`, `Add Existing`, and `Remove Project(s)` menu items for adding new or existing projects, or removing projects from the current project space. The `File View` tab of the `Project Window` displays a hierarchy of project spaces, the projects they contain, and each project's member files. The Version Control status of files is also provided.

Project Files

The project is defined in two files with the same name, but different extensions. The Hawk file extensions are `*.PJT` and `*.MPJ`. A backup of the project file (`*.mpjBackup`) is saved every time the project is saved.

A project file (`.PJT`) is essentially a configuration file and state file all in one. Workspaces are stored as additional "state files" within the project file. Once you understand the format of a Hawk configuration file, you understand a Hawk project file. These files look just like standard Windows `.INI` files with headings enclosed in square brackets, and statements on lines following these headings. Statements take the form of `<keyword>=<value>`.

The primary difference between a Hawk configuration or project file and a Windows `.INI` file, is that keywords may not be repeated within a section of a Windows `.INI` file. In a Hawk configuration or project file, keywords may be repeated. This is possible because Hawk processes these files directly, without going through Windows. For more information on the format of configuration and related files, refer to the topic [Configuration and State](#) in [Chapter 17 Configuration Files and Command Line Parameters](#) of this manual.

Project

A Hawk project is a single file containing a list of filenames and settings that are intended to make up the project. It contains a tree of components; however, no units can reside at the root level of the project. The project file containing the project's files and settings is stored in the directory in which it was created; it is given the name of the project with a `.PJT` extension.

Language, compiler, version control, clipboard and scrap, and other options can all be stored in projects. Storing settings with Hawk projects allows for unlimited feature-variation when going from project to project. With projects, the user can create mini-environments for sets of files that, for example, use different compiler options, have different promotion groups for version control, or use different languages. Closing one project and opening another automatically sets the options to that of the open project.

Overall, Hawk projects allow the user to organize a group of files as a unit giving quick access to the configurations necessary for using and working with those files more efficiently.

Workspace

A workspace is a separate instance of Hawk. It is a "mini-project" within a project. When you change workspaces, you have the option of picking up where you left off with a group of files, regardless of how long it has been since you worked with them.

A workspace differs from a project in that it does not store the system-wide options normally stored in a configuration file. In a sense, the workspace is like a swapped Hawk state file. A state file stores transitory information about the state Hawk was in when last exited. In addition, other state information, such as search options, response histories, bookmarks, etc. are stored as part of the project. State files retain information about the windows and buffers open during the last Hawk session, and the position they were in. Workspaces are like "mini-state" files within projects. They retain different sets of window information for each workspace.



For additional discussion of workspaces, refer to the topic [Creating, Selecting and Saving Workspaces](#), in this chapter.

Component

A component is analogous to a folder. It holds units as well as other components. Hawk recognizes certain component types, including the following:

- User State Program
- System State Program
- Descriptor
- Driver
- File Manager
- I-code library
- O-code library
- collection: This component type does not actually create any binary output. It is a holding place for other components and units; it is used to specify common property settings.

Component Options

The following are options accessed by selecting `Properties` in the component pop menu:

- `Name`: The name of the component.
- `Chip`: Override the chip type. If a chip different than the project chip is selected, all units and components contained within the selected component will use this chip type when compiling.

- **Type:** Change the component type.
- **Output File:** If the component generates a binary, this becomes the name of that binary.
- **Description:** Enter a simple description for this component. The description is not used in anyway. It's simply saved in the Project file for documentation purposes.
- **Unit Root Psect:** defaults to None.
- **Exclude from build:** This option is only displayed for Components and Units. If checked, the selected unit or component will be excluded from any build. If the user specifically chooses to build a unit or component with this option checked, he option will be ignored.

Build Order

The order in which components are compiled is in order from first to last in the Project Window. A child component is compiled before the parent. The child components may be ordered by one of the following methods:

- mouse interface

If a component (child) is dragged and dropped on another component (parent), it will be put at the end of list of child components.

- keyboard interface

Alt-Up Move component up one spot next to its siblings

Alt-Down Move component down one spot next to its siblings.

Alt-Left Move component to next higher level

Unit

A unit is the lowest element in the Hawk Project tree. A unit represents an individual file of any type; Hawk recognizes specific file types.

Unit Maintenance

The Unit Maintenance dialog is one of the most important in the Hawk Project Manager. Select this dialog by choosing **Unit Maintenance** (accessed by right-clicking on a project component). This dialog allows you to add, delete, and reorder units within a component (see the next topic on the [Project Preferences](#)). When the dialog is opened initially, the units in that component are displayed with bitmaps to the left. If you choose to add new units, they display without a bitmap. You can then reorder the units. In addition, you can press the up/down arrow while holding the **Alt** key You can also delete units using the **Delete** button.

Below is a table of various file types and their extensions.

Table 7-1. Units

File Type	Associated extension
C Source	*.c
C++ Source	*.cpp, *.cxx
Header	*.h, *.hpp, *.hxx, *.d
Descriptor	*.des
ROF	*.r
I-Code	*.i
I-Code Library	*.il
Assembly	*.a
O-Code	*.o
O-Code Library	*.l
Makefile	makefile., *.mak
Batch File	*.cmd, *.bat

Project Preferences

Below is a list of project preferences and questions concerning them:

- `Continue on Error`: When doing a build, should Hawk stop if an error is encountered?
- `Perform Animation`: Should Hawk highlight the unit or component currently being built?
- `Build Modified Units`: Should Hawk save any modified buffers before performing a build?
- `Automatically adjust orientation`: Should Hawk automatically adjust the orientation of the splitter in the `Project` Tab? Should Hawk change the orientation of the splitter to vertical if the length of the project tab is greater than the height?

Creating a Project Space

To create a project space, do the following:

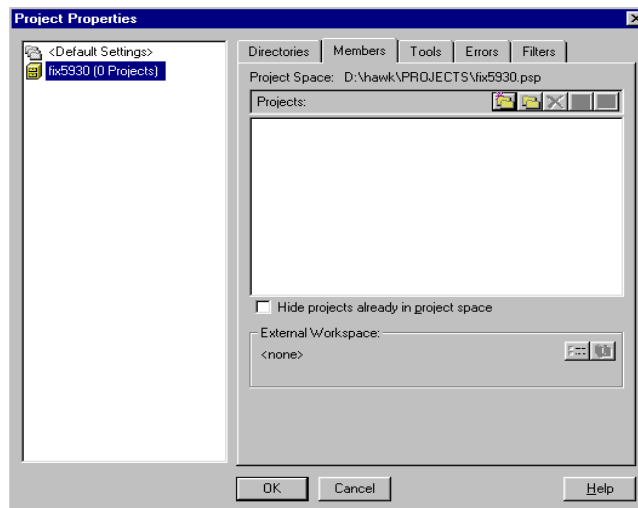
1. Click `Project` -> `Project Space` -> `New`.
2. Click `Browse` to find the appropriate directory for your new project space, or type a filename in the `Filename` box. Keep the following in mind when completing this step:
 - The `Look in same directory for external workspace` option automatically inserts the name of any similarly-titled Visual Studio Workspace (`.DSW`) file that may reside in the same directory in which the project space is being created in.

- A `.PSP` (Hawk project space) file type extension is automatically appended to the inserted name.
- Some of the contents of the corresponding `.DSW` file will subsequently be read and conformed, for use in creating the project space.
- The option supports Visual Studio 5.0 and 6.0 only. Turn off the option if neither of those versions of Visual Studio is being used, or if it is undesirable to have `.DSW` files converted to Hawk project spaces.
- For more information on what the `Look in same directory for external workspace` option does, read the topic [Reading External Makefiles and Workspaces](#), in this chapter.
- Hawk assumes the filename extension `.PSP` for project space configuration files if one is not supplied. You may specify another extension, if desired.

3. Click **OK**.

The project space is created and the Project Properties dialog appears, with the **Members** tab on top. The new project space is displayed in the list box on the left, as shown in [Figure 7-1](#).

Figure 7-1. Project -> Settings -> Members



Projects are added to project spaces, then files are added to projects, in the `Project -> Settings -> Members` dialog. Before covering that topic, however, it is necessary to have some understanding of the `Project Properties` list box and the process for setting project configurations.

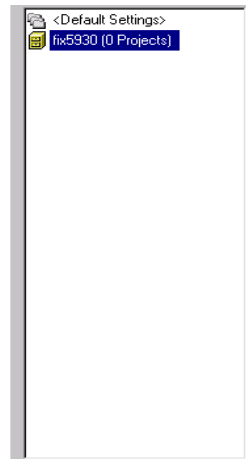
The Project Properties List Box and Project Settings

Hawk's `Project -> Settings` dialog is where many necessary Hawk configuration settings are made. One item in the dialog, the `Project Properties` list box, is important to all tabs of the dialog.

Project Properties List Box

The `Project Properties` list box (shown in [Figure 7-2](#)) is available from all tabs of the `Project -> Settings` dialog. It lists the current project space and any projects that the space contains. It also lists a special item called `<Default Settings>` that is used for setting global configurations within the dialog. Configuration settings can be manipulated for any item selected in the list box, but not all settings are available for all items. This is reflected by the fact that the `Project -> Settings` dialog changes depending on the item selected.

Figure 7-2. Project Properties List Box



Default Settings

Settings made in the project properties dialog while the `<Default Settings>` item is selected will apply to all projects until those settings have been changed for the projects individually. Settings made while a project is selected are called "project-specific settings" and are stored with that project only.

Certain items and tabs in the `Project -> Settings` dialog are only available when the `<Default Settings>` item is selected; others are only available when projects or project spaces are selected. For example, the `Members` tab of the `Project -> Settings` dialog is used for adding projects to spaces and/or files to projects; therefore, it is not available when the `<Default Settings>` item is selected (the `<Default Settings>` item does not store projects or files). The `Members` tab is the only tab that is not used for configuration settings and is therefore the only tab available at the project-space level (project spaces do not store configuration settings).

The other tabs in the dialog are available for the `<Default Settings>` item, as well as for selected projects; they are used for making configuration changes that can be either global (such as `Default Settings`) or specific to individual projects.

Working Directory

One important item to consider when setting project defaults is the project's working directory. The working directory is the directory from which all tool commands are run;

it is configured in the `Project -> Settings-> Directories` dialog. The default working directory is set to `%x`, a filename component macro that expands to the directory containing the project's configuration file.



More information on the filename component macros and the `Tools` tab can be found under the topic [The Tools Tab](#), in this chapter.

Once the project's desired default settings have been established, appropriate modifications can be made on a per-project basis. The information for setting project configurations and the tabs of the `Project -> Settings` dialog, is covered throughout the remainder of this chapter.

Setting Project Defaults

Configuration settings for new projects are inherited from the `<Default Settings>` item in Hawk's `Project Properties` dialog. This is the same dialog that displays when you first open a new project space. It can also be accessed by clicking on `Project -> Settings`.

Changes made to the settings while a project is selected are stored with that project only. Closing the project causes all settings to revert to the defaults. For this reason, it may be a good idea to set some configurations for the project before creating it so the settings remain unchanged when the project is closed.

To make default settings throughout the `Project Properties` dialog, highlight the `<Default Settings>` item in the `Project -> Settings` dialog, then make various configuration settings in the various tabs of the dialog. Once you have established the default settings, you can make modifications for future projects on a per-project basis.

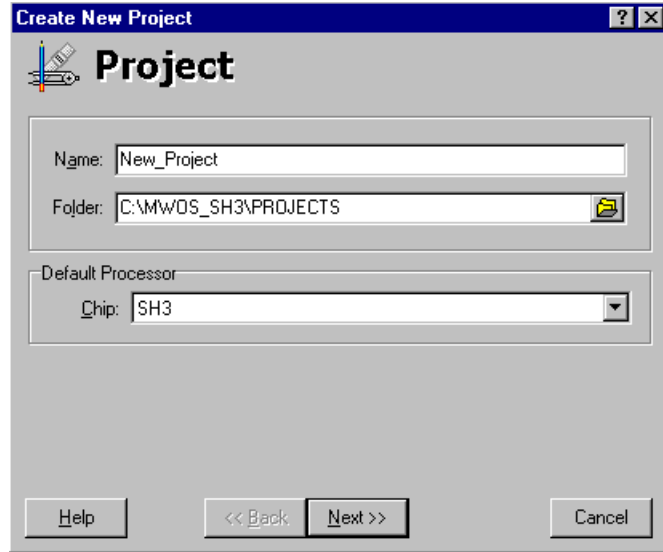
Creating and Modifying Projects

Once a project space has been created, individual projects can be created for the space. After creating a new project space, the `Project Properties` dialog appears. It is also possible to access the same dialog by clicking `Project -> Settings`.

However, we will not use the `Project Properties` dialog to add a project to the project space. The following steps will use a different method to create a Hawk project.

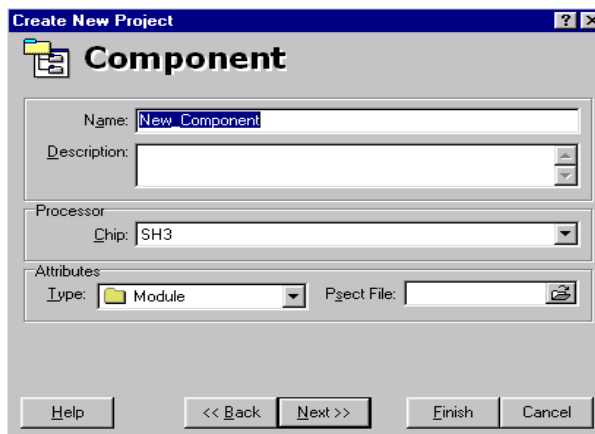
1. Click the `Cancel` button to dismiss the `Project Properties`
2. Click `Project -> Project Space -> Add New Project` to access the `Create New Project` dialog. The dialog shown in [Figure 7-3](#) appears:

Figure 7-3. Add New Project to Project Space



3. Enter the name of the project into the **Name** field. Also, make sure the path to the project's folder in the **Folder** field is correct and make sure the **Default Processor** displays the correct processor name.
4. Click **Next**. The **Component** dialog box appears, as shown in Figure 7-4.

Figure 7-4. Add New Component to Project



5. Projects consists of multiple components. Type the following into the component menu:

Name :

Description:

Chip: <Processor Name>

Type:

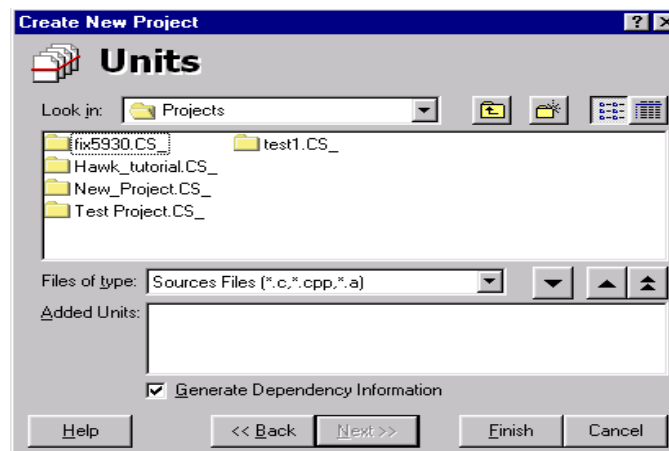
Psect File: <Browse to the appropriate directory>.

Complete the following fields:

Name:
 Folder:
 Chip: <processor name>

6. At this point, you can do one of two things:
 - If you would like to add units to your project, proceed to step 7.
 - If you do not want to add units to your project, click **Finish** and proceed to step 11.
7. Click **Next** to display the Units menu. Components consists of units, which can be library, header, or source files. The Units menu is shown in Figure 7-5.

Figure 7-5. Units Menu





8. Navigate to the file you want to add from the **Look in:** box.
9. To add the file to the **Added Units** display, you can either click the down arrow to the right of the **Files of type** box or simply double-click on the file. The file you chose in the full path list should now appear in the **Added Units** box.
10. Repeat step 9 until you are through adding files. When you are done, click **Finish** at the bottom of the window. The **Components** frame should display a folder with the name of your added component.
11. To save the project, select **Project -> Save**.


Hawk creates a new project configuration file in the chosen directory that uses the name typed in the **Add New Project to Project Space** dialog with a **.PJT** extension appended. The new project is listed in the **Project Properties list box**, under the current project space.

Hawk creates a new configuration file in the chosen directory with the name that was typed and a **.PJT** extension.

Adding Files to a Project

Files can be added once the project has been created. Files are added in the **Project -> Settings -> Members** dialog. To add files to your project, complete the following steps:

1. in the `Project Properties` list box, highlight the project to which you are adding files.
2. Add files using one of the following four methods:
 - Click the `Add New File`  button.
 - Click the `Scan`  button. Add individual files by typing in their names, or add multiple files using file filters (such as `*.C`).

Within the `Scan` dialog, use the `Browse`  button to browse for files. Add individual files by typing in their names.

Matching files in subdirectories are included as a result of the `Include Subdirectories` check box in the scan dialog.

 - Use the `External Makefile` option to read files from an existing makefile into the project being created.
 - Use the `VCS Project` option to "read" a version-control project file. (See [Using Version Control in Hawk](#), in [Chapter 9 Version Control](#).)
3. Click `OK` after completing one of the above options. Any matched files should be immediately added to the project.


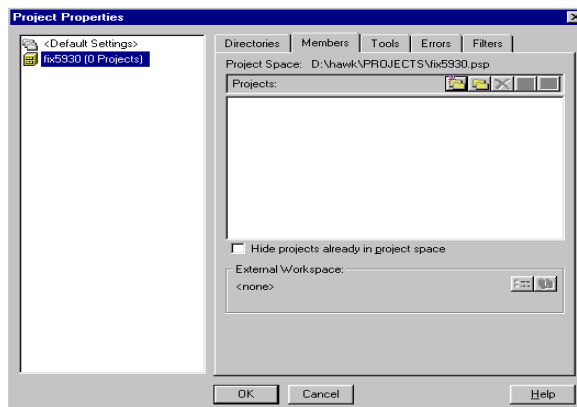
Files that are included in the project are listed in the `Files` list box in the center of the dialog. To remove files from the project, select the `Remove selected files from project`  button under the `Members` tab, shown in [Figure 7-6](#).

Figure 7-6. Project -> Settings -> Members

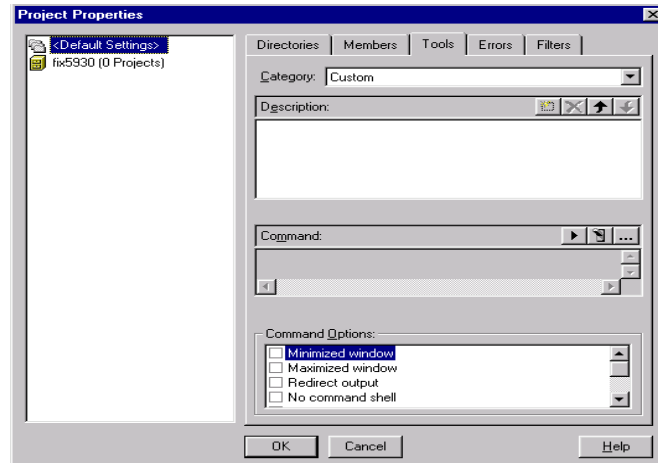


Project Tools

When the necessary files have been added in the `Members` tab of the `Project Properties` dialog, click on the `Tools` tab. The `Tools` tab is used for setting up various tools, such as command line compilers and version control utilities needed to perform various jobs on files being edited in Hawk.

The commands set on the `Tools` tab are the commands that are run when the various `Compile`, `Build`, `Debug`, etc., buttons and menu items are clicked. [Figure 7-7](#) shows the `Tools` Dialog.

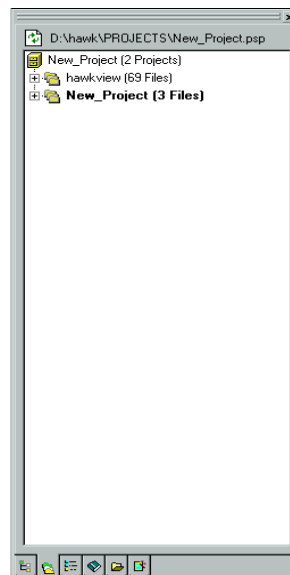
Figure 7-7. Project -> Settings -> Tools



Once the appropriate tools have been set on the `Tools` tab, click `OK`. The dialog closes and the project is created with the specified characteristics. Clicking on the project tab in the project window will now display the items in the project space.

Click on the `File View` tab of the `Project` window to display the project space with the included projects beneath.

Figure 7-8. Project Window: File View



To load project files for editing, double-click on them in the window, or click on the `Load Files` item in the `Project` menu.



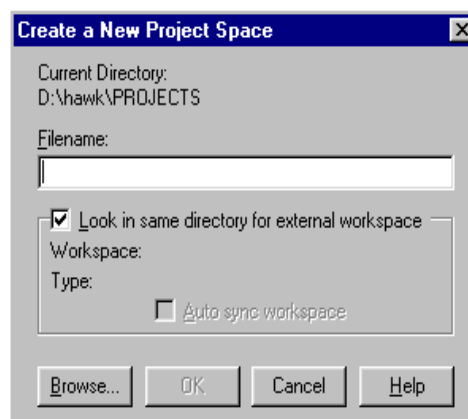
A `*.bat` or `*.cmd` file can be included in a Hawk project and used to run `os9merge`, `fixmod`, or other applications and utilities. To ensure that this batch file runs correctly after a module is created, your project should have a subcomponent that contains the module's source code in the component with the `*.bat` or `*.cmd` file.

Adding New Projects Using the Members Tab

To add projects from the `Members` tab of the `Project Properties` dialog using one of the following methods:

- From the `Add New Project to Project Space` dialog (shown in [Figure 7-9](#)), browse for individual project files or use extended selection list box rules (for example, `SHIFT/CTRL` keys with mouse clicks) to add multiple projects from the same directory at a time.

Figure 7-9. Add Projects Dialog



- From the `External Workspace` dialog at the bottom of the `Members` dialog, select the `Setup external workspace` button. Hawk uses this to read projects. The projects will be read from the Visual Studio Workspace into the current Hawk project space. See the topic [Reading External Makefiles and Workspaces](#) for more information.


Any matched projects should be immediately added to the project space. The projects are shown below the current project space in the `Project Properties` list box.

Reading External Makefiles and Workspaces

Hawk can extract the names of projects or files from either Visual Studio Workspaces or specified makefiles, adding them to the list of project members or project spaces, respectively. This little bit of automation helps keep the Hawk projects and project spaces synchronized with the IDE. Hawk does not use the Workspaces and makefiles directly, but adds the extracted filenames to Hawk projects (`.PJT` files) and project spaces (`.PSP` files).


To have Hawk automatically use this capability, mark the **Look in directory for external makefile/Workspace** option in the Add New Project to Project Space dialog, or in the Add New Project to Project Space dialog. Hawk automatically searches the directory (in which the project/project space is being created) for any makefiles or Workspace files whose names match the project/project space name specified. If the files are found, Hawk automatically inserts the name of the file into the Add New Project to Project Space/Project Space dialog. The files or projects that are part of the Workspace or makefile are subsequently read into the Hawk project or project space.

To have Hawk manually read files or projects from selected makefiles or Workspaces into Hawk projects or project spaces, complete the following:

1. To read a Workspace, select the current project space in Hawk's **Project Properties list box** in the **Project Properties** dialog under the **Members** tab. To read a makefile, select a project from the same list box.
2. Select the **Setup**  button to bring up the **Read User Makefile** or **Select the type of parser** from the **Makefile/Workspace type** combo drop-downs to use for parsing the filenames from the respective file. Keep the following in mind:
 - A parser is composed of a regular expression pattern used to search for filenames and other pertinent strings in the makefile.
 - If none of the predefined Workspace or makefile types match the makefile being used, a custom parser can be defined that will "understand" the Workspace/makefile type.
 - Existing parsers can be used as references for building custom parsers. (For detailed steps on making parsers, see the next topic on [Steps for Setting up New Makefile/Workspace Parsers.](#))
3. Press **OK**.
4. In the **Project -> Settings -> Members** dialog, press the **Read** button. The project or project space will subsequently be populated with files or projects that correspond to files or projects contained in the associated makefile or Workspace.

Steps for Setting up New Makefile/Workspace Parsers

Use the following steps to set up a new makefile/Workspace parser:

1. Make sure you are in the **Project Properties** dialog box by selecting **Project -> Settings**. Select the **Members** tab.
2. Highlight a project or a project space.
3. Click the **Setup external workspace** button  in the **External Makefile** section.
4. Browse for the appropriate makefile or Workspace file using the **Browse** button next to the **Makefile Name** edit box in the **Read User Makefile** dialog.

5. In the `Makefile Type` edit box, give the parser a descriptive name.
6. Enter an applicable regular expression in the `Regular Expression Parser` edit box.



Creating a new makefile or Workspace parser requires some knowledge of regular expressions. More guidance for creating regular expressions is provided under the topic [Regular Expressions](#) in [Chapter 10 Search and Replace and Other Navigational Tools](#).

7. Click **OK**.
8. Click the **Read external makefile** button.
9. Click **OK**. The respective list boxes in `Project Properties` should contain the files or projects used by the makefile.

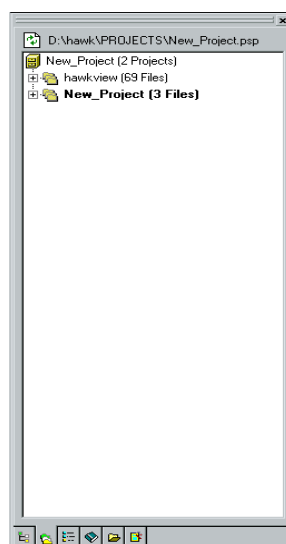
Characteristics of the File View Tab of the Project Window

The `File View` tab of the `Project Window` is specifically intended to display projects and project spaces. By default, the `Project Window` is displayed on the left-hand side of the Hawk screen.






Projects, project spaces, and files will only show in the `File View` tab of the `Project Window` when a project space containing those projects and files has been opened in Hawk. Projects are displayed under the project space, and project member-files are then grouped under each individual project by filters that are established in the `Project -> Settings -> Filters` dialog. Each filter is displayed as a folder containing files of the appropriate extensions. (Refer to the topic [The Filters Tab](#), in this chapter, for more information on project filters).

To open a file for editing, double-click on the file in the `File View` window (shown in [Figure 7-10](#)).


Figure 7-10. Project Window: File View



When files are under version control, file-icons depicting different states of the file are shown in the `Project Window`:

-  When the files are checked-in (or read-only) the icons that represent the files are grayed out.
-  When the files are checked out, Hawk conveniently places a red check mark to the upper right of the file icons.
-  If another user has a lock on a file that is part of the project being worked on, Hawk will display the file with a gray check mark in the `Project Window`.
-  Some Version Control projects are set to delete work files when they are checked in. Hawk displays files not found at the specified location (usually archived) with an exclamation point.
-  When the current edit buffer differs from the latest revision in version control, Hawk gives the corresponding file icon a slightly different color.



Press the refresh button  from time to time when using the `Project Window` to carry out version control operations, or view version control status, to see the most current status of the files.

The primary purpose of the `Project Window` is to open files and facilitate version control operations. Right-click on a file, or one of a selected group of files, in the `Project Window` to bring up a menu listing various operations that can be performed on the file(s). Additional right-click popup menus are available when right clicking on a project or project space.

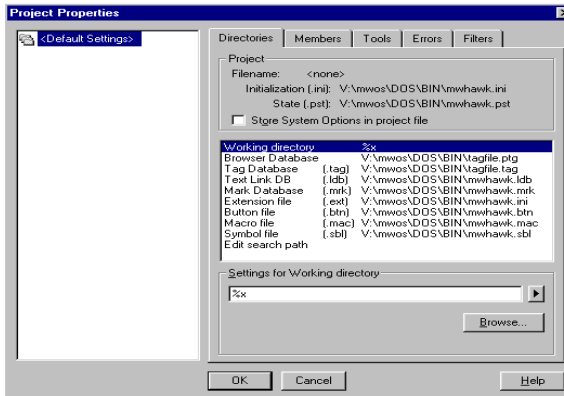
The Directories Tab of the Project Properties Dialog

The `Directories` tab of the `Project -> Settings` dialog is used to display and set the project's working directory as well as other directories for various system files used in Hawk. Complete the following steps to access/edit the appropriate version of the `Directories` tab:

1. Access the `Project -> Settings` dialog.
2. In the `Project Properties` list box:
 - Select a project if the directories being viewed or modified are intended to be specific to the selected project only.
 - Choose `<Default Settings>` if the directories being viewed or modified are intended to be global.
3. Click on the `Directories` tab (shown in [Figure 7-11](#)). Most of the directories and filenames listed in this dialog can be made project-specific, but only if the `Store System Options in project file` check box is marked. Once it is

marked, the names of most of the listed files will be available for editing. They can then be made project-specific by giving them names that are unique for the selected project.

Figure 7-11. Project -> Settings -> Directories



A brief description of each of the directory and file listings in Project -> Settings -> Directories follows:

- The **Working Directory** is the directory from which commands set on the **Tools** tab of the Project -> Settings dialog will be run. By default, the working directory is set to %x, a filename component macro that expands to the directory in which the project file resides.
- The **Browser Database** indicates the name and location of the Microsoft Browser Database (.BSC) or the compiled tags database (.PTG). More information about tags can be found in [Chapter 10 Search and Replace and Other Navigational Tools](#).
- The **Tag Database** indicates the name and location of the file that contains Tags information.
- The **Text Link DB** indicates the name and location of the file that stores Button Link information. More information about Button Links can be found in [Chapter 10 Search and Replace and Other Navigational Tools](#).
- The **Mark Database** indicates the name and location of the file that stores bookmark information.
- The **Extension File** indicates the name and location of the file that stores extension-specific information. In the Settings area, specify a **User Named File**, the **Configuration file**, or the **Project File**.
- The **Button File** indicates the name and location of the file that stores toolbar resources and default button descriptions. Refer to the online topic *Dockable Toolbars and Windows* in Hawk's *Frequently Asked Questions*.
- The **Macro File** indicates the name and location of the file that stores API and keystroke macros. More information about API macros can be found in [Chapter 15 Extending Hawk](#)

- The `Symbol File` indicates the name and location of the file that stores information about symbols. If you create a project within a project space, the default symbol file name is the name of the project with a `.SBL` extension (such as `<project_name>.SBL`). If an existing project's symbol file is `MWHAWK.SBL` (the non-project default name), you will be prompted to change the symbol file name. It is advantageous to have project-specific symbol files in order to limit the size of the files, thereby optimizing Hawk's performance.

Selecting the `Symbol File` option in the `Directories` tab makes two options available: `Auto-Update Symbols database` and `Show Definitions in Symbols Tab`. These options control whether or not symbols information is automatically computed when projects are made current. Unmarking these options can speed up the process of opening projects. More information on Symbols can be found in the chapter [Chapter 10 Search and Replace and Other Navigational Tools](#).

- The `Edit Search Path` specifies the order of directories to be searched when opening files without specifying the directory in which the file resides. The `Edit Search Path` is primarily useful when using Hawk's status line prompt for opening files, instead of common dialogs. To define more than one directory in the `Edit Search Path`, separate the directories with semicolons. Add an asterisk to the end of a component of the `EditPath` to specify an entire directory sub-tree.

Example

The `EditPath c:\include\msvc;c:\cw*` will cause Hawk to first look in `c:\include\msvc` and then in `c:\cw` and then in all subdirectories of `c:\cw`.

The traversal of the sub-tree is done depth-first, meaning that a particular sub-tree is traversed all the way down to its leaves before moving on to the next sub-tree.



The backslash prior to the asterisk is optional; the two forms are entirely equivalent.

Hawk projects can store configuration settings individually, making them a convenient way to have multiple sets of configurations in one editor. Some settings are stored automatically, when changes are made to the `Project -> Settings` dialog while a project is selected. Others will only be stored with the project if certain options are marked in various Hawk dialogs. The settings that may be stored with projects are listed below:

- System Options: Mark `Store System Options in project file` for selected projects in `Project -> Settings -> Directories`.
- Auto-save options: Mark `Store Auto-save Options in project file` in `Customize -> Environment -> Backup` while the desired project is open.
- Language options: Select the `Project File` radio button when the `Extension File` option is chosen from the list in `Project -> Settings -> Directories` for selected projects.

- Filename Filters: Mark **Save Filename Filters in project file** in **File -> Filters** while the desired project is open.
- Clipboard/Scrap options: Mark **Store Clipboard/Scrap in project file** in **Customize -> Environment -> Clipboard** while the desired project is open.
- Compiler and Command Line Version Control settings: Make the desired configurations in **Project -> Settings -> Tools** while the desired project is selected.

In addition to indicating that certain system options be stored with the project, the **Store System Options in project file** check box in **Project -> Settings -> Directories** allows the configuration file names listed to be edited. Changing the names of those files on a per-project basis allows them to be made project-specific so that the settings they store will also be project specific. To set the **Store System Options in project file** check box on the appropriate version of the **Directories** tab, complete the following steps:

1. Access the **Project -> Settings** dialog.
2. Complete the following in the **Project Properties** list box:
 - Select a project if the options being viewed or modified are intended to be specific to the selected project only,
 - OR -
 - Choose **<Default Settings>** if the options being viewed or modified are intended to be global.
3. Click the **Directories** tab.
4. Check the **Store Systems Options in project file** check box, and press **OK**.

Reading Configuration Settings from Other Files

In addition to storing configuration settings with projects, it may sometimes be desirable to read specified configuration settings from designated configuration files into a project or into the global configuration settings. It is possible to have Hawk read configuration settings from a configuration file using a dialog on the **Customize** menu called **Read Configuration Data from a File**. It is accessed by selecting **Customize -> Read configuration Data**. The configuration information gathered will be placed in Hawk's main configuration file or in a Hawk project file, depending on the data being read as well as Hawk's current state (i.e. whether a project is open or not).

The series of check boxes in the dialog limit what settings are read from the file. The **Update Configuration Files** check box immediately updates the project and/or configuration files with the specified information when marked.

The Tools Tab

The **Tools** tab of the **Project -> Properties** dialog is used for setting up various tools, such as command line compilers and version control utilities needed to

perform various jobs on files being edited in Hawk. These tools can be made project-specific or global. The `Tools` tab is located in `Project -> Settings`.

Tool Categories

The items in the `Tools` tab change depending on the tool category. Different tool categories can be selected from the drop-down list under the `Category` combo box. Project tools include command line entries for compile utilities, make or build utilities, executables that may be run on a regular basis, and command line version control utilities. Hawk does not come with its own utilities (such as compilers and version control); these must be purchased separately. Hawk simply provides a way to use the utilities in a more practical and efficient way.

The tools in the various categories correspond to menu items and buttons in Hawk that are used to run the utilities. When one of the menu items or buttons is clicked, Hawk runs the tool by shelling to DOS and running the tool's command. Any output generated is captured for display in designated tabs of Hawk's `Output Window`. The specific tool categories include the following:

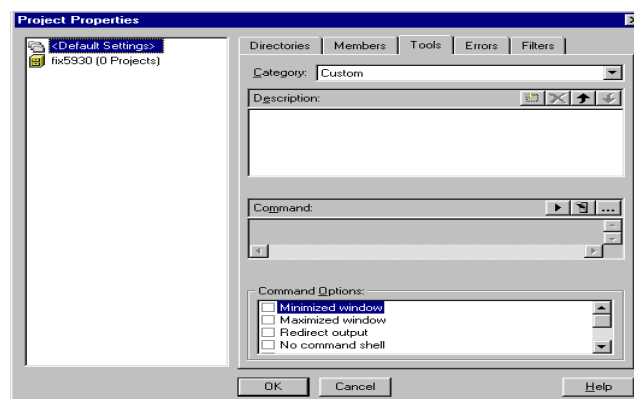
- [Custom Tools](#)
- [VCS Tools](#)

The following sections describe the tool categories in the `Project Properties` dialog box `Tools` tab.

Custom Tools


The `Custom` category of tools is provided for running any DOS program from within Hawk. Adding a custom tool in `Project -> Settings -> Tools` will add it to a list at the bottom of Hawk's `Tools` pull-down. Clicking on a tool's name in the menu will then run the tool's associated program. Two custom tools are already set up in Hawk: Windows's `Paint` program, and the `API Database Editor`, a tool for editing Hawk's `API Assistant` databases. The `Custom` category of the `Project Properties` dialog is shown in [Figure 7-12](#).

Figure 7-12. Project Properties Dialog: Custom Category



To make a new custom tool, complete the following steps:

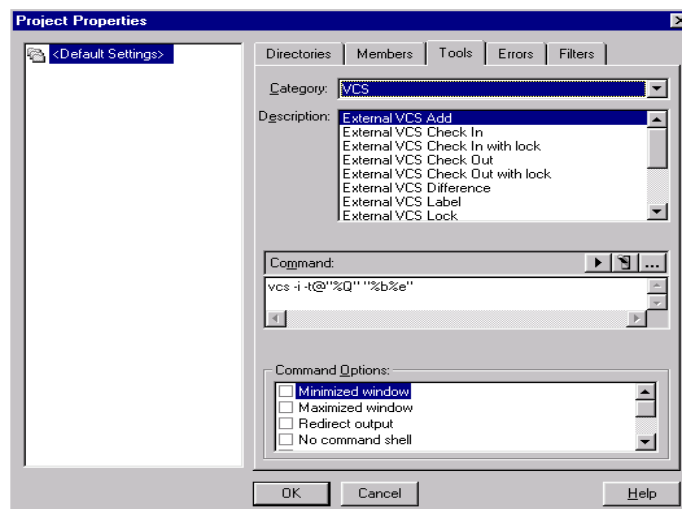
1. Select `Project -> Settings -> Tools`.

2. Choose **Custom** from the list of categories in the list under the **Categories** combo box.
3. Click the **New**  radio button.
4. Give the custom tool a name.
5. Click **OK**.
6. Highlight the new custom tool in the list box.
7. In **Command**, edit the box and type the appropriate command for running whatever program the tool is intended for (to make a tool that runs a Hawk API, type the API in the **Command** box, and precede it with an **!**).
8. Choose any necessary command options. For descriptions on the command options, see the topic [Command Options on the Tools Tab](#), in this chapter.

VCS Tools

The VCS category of tools in the **Project -> Settings -> Tools** dialog (shown in [Figure 7-13](#)) consists of a number of tools associated with various version control operations; these operations are available on different menus, buttons and dialogs in Hawk. Set the tools with the commands that will run any command-line version control utilities used to perform version control operations (such as `GET.EXE` or `PUT.EXE`).

Figure 7-13. Project Properties Dialog – VCS Category



You will notice that the VCS tools come pre-configured with commands for some of the more popular version control systems. The pre-configured commands change depending on the provider-name chosen in the **Command Line Provider** list box in the **Tools -> Version Control -> Setup** dialog.

You can use filename component and symbolic macros as placeholders to insert or transform filenames and filename components (such as directories and filenames) used with the version control utilities. This obviates the need to manually insert filenames every time the files being edited or checked in/out change. More information

on filename component and symbolic macros can be found in the sections on [Filename Component Macros](#) and [Symbolic Macros](#).

You should only be concerned with the VCS tools if you use command line (DOS) version control utilities, such as `PVCS GET.EXE` or `MS VSS SS.EXE`, to carry out your version control operations. You should also be aware that the VCS category of tools is not the only method for using version control in Hawk. Version control operations can also be performed via SCC API integration. More information about using either of the Version Control integration methods can be found in the chapter on [Chapter 9 Version Control](#).

Setting up Project Tools

Setting up most of the project tools is a fairly straightforward process. In their simplest form, the steps for configuring project tools are as follows:

1. Choose the appropriate tool-category.
2. Highlight or choose the tool to be configured.

Enter the appropriate command in the `Command:` edit box at the bottom of the dialog (to make a tool that runs a Hawk API, type the API in the `Command` box, and precede it with an `!`).



The preceding steps are based on the assumption that the utility being launched runs successfully from any directory in the operating system. This means that the proper environment variables and `PATH` statements must be set according to the instructions provided with the utility in question. Otherwise, the full path and filename of the utilities used must be inserted with the command when performing Step 3.

You may notice that many of the project tools in the `Tools` tab are already configured. These pre-set commands may not apply to the utilities actually being used. In such cases, the tools will need to be reconfigured.

Example

Highlighting the `Custom Tool` for the first time shows the following command in the `Command:` edit box:

```
cwedit.exe
```

Command Options on the Tools Tab

Each category of tools in the `Project -> Settings -> Tools` dialog has a number of options that affect what happens when any of the various tools are run.

Example

To control how Hawk redirects the output generated by any of the utilities associated with the tool being used, one would mark or unmark the `Redirect output` option in the list of command options displayed underneath `Command` edit box.

The command options should be set individually for each tool being used, regardless of the tool's category. Descriptions of the command options follow:

- `Minimized window` **Starts the tool with its window minimized.**
- `Maximized window` **Starts with tool with its window maximized.**
- `Redirect output` appends the following items to the end of the Command that runs the Tools program:
 - A DOS redirection operator.
 - The name of the Error File listed on the `Errors` tab.

With this option marked, any output that normally displays in a DOS window will be redirected into the error file. Hawk then uses an FTEE program to display that information in the `Build` tab of the `Output Window`.

- `No Command Shell` causes the command not to be run in a command shell (`COMMAND.COM` or `CMD.EXE`). If you select this option, redirection and other services will not be available. If you are executing a Windows program, there is no need for a command shell.
- `Save Current File` causes the current file to be saved before the corresponding tool runs.
- `Save All Files` causes all files opened in Hawk to be saved before the corresponding tool runs.
- `Background` determines if the program is executed in the foreground or the background. If you elect to execute the program in the background, a message will appear in Hawk's status bar, and a beep sounds to indicate that the program has been completed.
- `Use VDOS` causes output from the compile or build to be sent to the VDOS window. This is convenient for cutting and pasting from the output, and for re-executing commands, but you may not be able to see a prompt if the program prompts for a response.
- `Prompt for Arguments` causes a prompt to come up before the tool executes, allowing the user to insert extra parameters to be used by the associated program.

Filename Component Macros

When compiling or building projects, it is usually necessary to have a source or make-file for the target of the build or compile. When setting up target files for project tools, it isn't convenient to use literal filenames, since the files will most likely vary. Hawk provides a number of `Filename macro hints` to be used on the various project-tool command lines, which will expand the names, or portions of names of current files or projects in Hawk.

Each category of tools has one or more macro assistants that can be used to quickly insert appropriate filename component macros at the cursor-position of any one of the tools' command lines. The macro assistant provides a pop-up menu listing available

macros. To access the pop-up menu, click the black right-arrow on the gray portion of the Command edit box.



A sample macro assistant pop-up menu is provided in [Figure 7-14](#).

Figure 7-14. Filename macro hints

Volume	%v
Path	%p
Root	%r
Directory	%d
Extension	%e
Base name	%b
Project path	%x
Project root	%y
Percent Sign	%%
Macro expansion	\${name}

A list of these macros and the filename specs they represent are listed in [Table 7-2](#). An example of how each macro looks when expanded is also provided. The examples are based on a file named `FOO.C` residing in the directory `c:\test` with the Hawk project `FOOPROJ.PJT` open (also residing in `c:\test`). A version control project `VCSproj` is open as well, with its project file also in `c:\test`.

Table 7-2. Filename Component Macros

Components	Description
%v	Volume The drive component of the workfile, consisting of a letter and a colon. Example: C:
%p	Path The path component of the workfile. This component has no filename and no drive component. It has a trailing backslash. Example: \TEST\
%r	Root The filename of the workfile, less the extension. Example: FOO
%d	Directory The directory component of the workfile. This component has no filename and no drive component. It also has no trailing backslash. Example: \TEST

Table 7-2. Filename Component Macros

%e	Extension The extension of the workfile. This component begins with a dot (.) unless the extension is null. Example: .C
%b	Basename The complete workfile name, less the extension. Example: C:\TEST\FOO
%x	Project Path The path component of the project file currently in use (usually indicates the project directory). Example: C:\TEST
%y	Project Root The base filename of the project file currently in use. Example: FOOPROJ
%%	Percent sign Since component macros begin with a percent sign, you must use two percent signs to represent a literal % in your command.
\$(name)	Macro expansion This does something really cool, just wait and see.

Symbolic Macros

Symbolic macros are another form of macro that can be used with many of the commands in the `Project -> Settings -> Tools` dialog. Hawk's Symbolic macros are meant to make the process of constructing command lines more flexible and convenient. A symbolic macro that is commonly used with some of the default commands is `$(FTEE)`, which expands the full path and filename of `FTEE.EXE`, a utility that pipes output to Hawk's `Output` window. A list of Component Macros and their descriptions appear in [Table 7-3](#).

Table 7-3. Filename Component Macros

Component	Description
\$(BROWSEFILE)	Browser file as defined in the <code>Directories</code> tab of the <code>Project -> Settings</code> dialog.
\$(FTEE)	Starbase's command for splitting output so that it goes to a file and to the screen. This command varies according to platform.
\$(HOME)	Hawk's home directory.

Table 7-3. Filename Component Macros

\$(KEYWORD)	The word at the cursor when a context-sensitive help function is invoked. This is used in the Default Help file settings. (Help menu <code>Configure Index...</code> item)
\$(TAGFILE)	Tag file name as defined in <code>Project -> Properties</code> .
\$(USER)	User ID defined in <code>Tools -> Version Control -> Setup</code> .
\$(VCSLABEL)	Revision label as defined in <code>Tools -> Version Control -> Maintenance</code> .
\$(WTAGS)	Starbase's command for generating a tags database. This command varies according to platform.

Braces {} may be used instead of parentheses ().

Additional symbolic names can be defined with the `SetStringMacro` function (help for the `SetStringMacro` function and any other Hawk API can be found in Hawk's online help). Any names not defined are taken as environment variables.

The Errors Tab of Project Properties

The `Errors` tab of the `Project Properties` dialog is used for specifying the name of the file used to capture output produced from a `Build` or `Compile`. It is also where appropriate error parsers are specified for accessing the errors displayed in Hawk's `Build` window (a tab on the `Output Window`).

Hawk uses two methods for capturing and displaying output from a compile—`FTEE` and `VDOS`. Once the output has been captured, Hawk uses `Error Parsers` to access the files that contain the errors in that output.

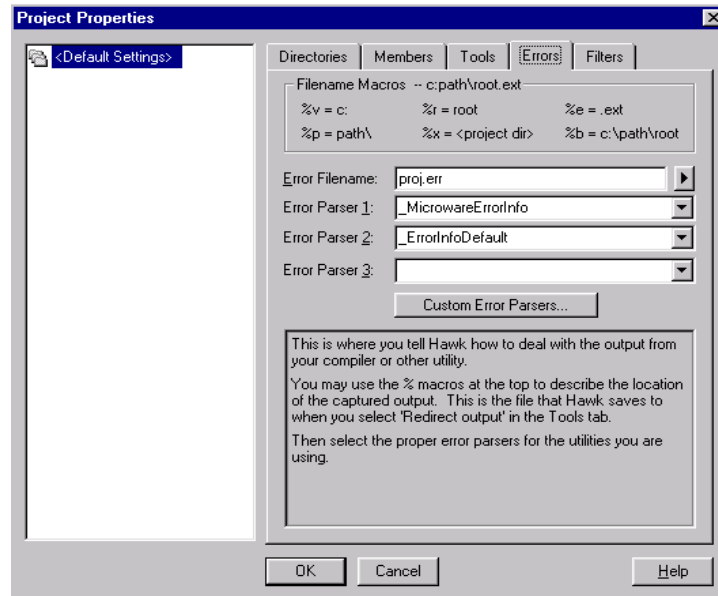
An error parser is a function that takes error messages from a compiler, assembler, or linker and extracts the name, line number and error message for the file in which the error occurs. Hawk uses that extracted information for error navigation. The error parsers allow you to double-click on an error or warning in the `Output Window` in order to bring up the source file(s) containing the error(s). The cursor will be positioned at the line where the error or warning occurs.

To access/edit the appropriate version of the `Errors` tab, complete the following steps:

1. Access the `Project -> Settings` dialog.
 2. In the `Project Properties` list box:
 - Select a project if the error configurations being viewed or modified are intended to be specific to the selected project only.
- OR -

- Choose `<Default Settings>` if the error configurations being viewed or modified are intended to be global.
3. Click the **Errors** tab (shown in [Figure 7-15](#)).

Figure 7-15. Project -> Settings -> Errors Tab



There are three combo-boxes in the **Errors** tab and one edit box for the Error File. Complete the boxes as follows:

- In the edit box `Error Filename:`, specify the file needed to capture any output produced by a compile or build. The Error File is the name of the file that is used to capture the output of commands that have the `Redirect output` command option checked. (See the topic [Command Options on the Tools Tab](#), in this chapter, for more information).
 - Error files are created in the current directory.
 - The error file is set to `PROJ.ERR` by default, but can be changed to any filename desired.
 - Filename component macros can be used for the filename, but it's usually best to leave it as it is.

The output captured by the error file will be used to display in the `Build` tab of Hawk's `Output Window`.

- Use the next three combo-boxes to specify any error parsers necessary for accessing the errors in the `Build` window. Only one parser is necessary, but up to three may be used.

Example

The output of a build or rebuild might contain output from a compiler, linker and assembler. If so, three error parsers will process the errors more efficiently than one.



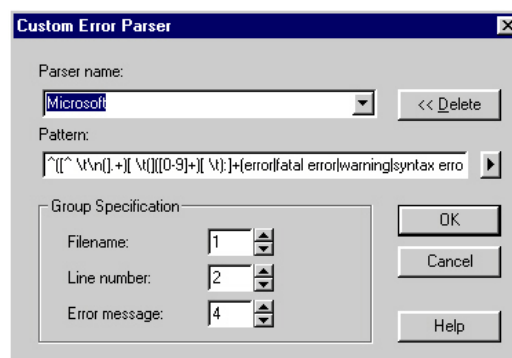
Do not be too concerned if an error parser for your compiler is not listed. When in doubt, try the Default parser ([DefaultErrorInfo](#)).

- The drop-down lists under the error parser combos contain the names of various error parsers available to Hawk. More parsers can be added by marking the **Error Parsers** options in Hawk's **Tools -> Customize -> Libraries** dialog.
- Choosing the appropriate error parser depends on the compiler being used. Usually the correct parser contains the name of the compiler or the name of the company that produces the compiler.
- If there are no parsers for a particular compiler, it is possible to make a custom error parser. Press the **Custom Error Parsers** button to access a dialog in which custom error parsers can be built, as described in the next section, Custom Error Parsers.

Custom Error Parsers

The error parsers available for Hawk are listed in the drop-down lists under each of the three error parser combo boxes in **Project -> Settings -> Errors**. More error parsers can be added to the list by clicking the **Error Parsers** options in the list of Hawk **Libraries** in the **Customize -> Libraries** dialog. If the appropriate error parser is not available, it is possible to make one using the **Custom Error Parsers** dialog (shown in [Figure 7-16](#)), accessed by clicking the **Custom Error Parsers** button in the **Errors** tab of the **Project Properties** dialog.

Figure 7-16. Custom Error Parser Dialog



All error parsers are made up of regular expression patterns. These patterns are used for searching error output for the filename, line number and error message of the errors that were produced by the compiler. To make a custom error parser, you must create a regular expression that will find the output strings that make up the errors. Use regular expression grouping and alternation patterns to group the expression pattern in a way that the filename, line number, and error message of the matching strings will each be made into separate entities.

Example

A sample regular expression for making an error parser for the MS J++ Java compiler is shown in [Table 7-4](#). Instructions for making the parser are also provided.

Table 7-4. Error Parser for the MS J++ Java Compiler

Description	Regex Pattern	Groups
Microsoft J++ Java	<code>^(.+):([0-9]+):[\t]*(.+)\$</code>	Filename:1
		Line Number:2
		Message:3

To make a custom error parser using the above example, complete the following:

1. Go to the `Custom Error Parser` dialog and copy the name of the above parser into the `Parser Name` field.
2. Copy the regex pattern into the `Pattern` field.
3. Fill in the appropriate group information.
4. Select `OK` in the dialog. The parser will then be listed as `_ErrorInfoParser MicrosoftJ++Java` in the `Project -> Settings -> Errors` dialog.
5. Choose `_ErrorInfoParser MicrosoftJ++Java` from the drop-down lists under the `Error Parser` combos.
6. Click `OK` in the `Project -> Settings -> Errors` dialog.

It is important to remember that syntax elements of strings that make up error output for any compiler vary depending on the compiler being used. A unique regular expression must usually be made for the output of each compiler.

Traversing the Output

You can use `Search -> Find Next Error` to go on to the next error in the `Output Window`. Some keymaps have keystroke equivalents to these commands. There is also a button available on the `Standard` toolbar to go to the next error.

[Table 7-5](#) displays some keystrokes that belong to the CUA keymap that facilitate movement in the `Output Window`.

Table 7-5. CUA Keystrokes

Keystroke	Action
SCZ	Bring up output window.
X	Return to current document.

Table 7-5. CUA Keystrokes

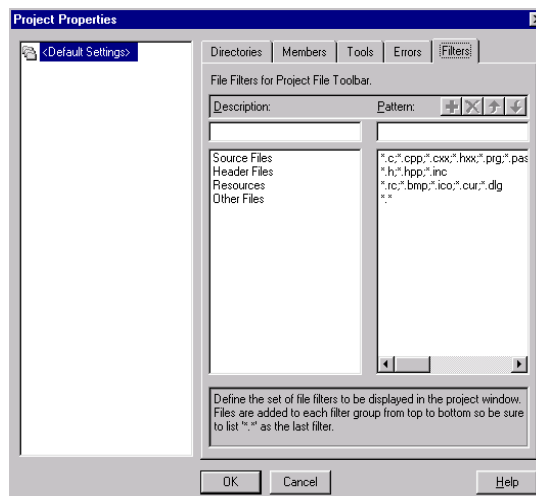
SCY	Parse next error.
C [Home]	Go to first line.
C [End]	Go to last line.
CT	Go to next tab stop.
SCT	Go to previous tab stop.

The Filters Tab

The `Filters` tab of the `Project Properties` dialog is used to specify how files should be sorted and displayed in the `File View` tab of the `Project Window` (the `File View` tab is described in the topic on [Characteristics of the File View Tab of the Project Window](#), in this chapter). The `Filters` tab (shown in [Figure 7-17](#)) controls how project files are "grouped" for display in the `Project Window`. The grouping is accomplished using wildcard characters and filename extensions.

The `Filters` tab is located in `Project -> Settings`.

Figure 7-17. Project -> Settings -> Filters Dialog



The default file filters for project files are Source Files, Header Files, Resources, and Other Files. Use the `Filters` dialog to add, remove or rearrange filters in the list. To add a new filter, complete the following steps:

1. Give the new filter a `Description` and a `Pattern`, for example "All Files" and *.*.
2. Click **Add**.

The new filter will be added to the top of the list.



Be careful not to put all-encompassing filters, such as `*.*`, at the top of the list. The filters sift through the files in the order that they appear in the **Project -> Settings -> Filters** dialog. Any files filtered by the first filter won't show up under subsequent filters. Therefore, if `*.*` is listed as the first filter, none of the other filters will appear to work.

Project Setup Checklist

Now that you have created your project and you are familiar with the various tabs of the **Project -> Settings** dialog, it is a good idea to review the settings to make sure they are appropriate for the project. A quick once-over will avoid surprises later. There are several dialogs to revisit.

1. The **Project -> Settings -> Directories** dialog:

Check the **Working Directory**. Be sure to define a working directory for your project. An explicit directory or `%x` (the path of the project file) are usually the best choices. Remember that the items on the **Directories** tab are only available if a project or the `<Default Settings>` item is chosen in the **Project Properties** list box.

2. The **Project -> Settings -> Tools** tab:

Check the **Project Command Lines**. Make sure the major project-wide command lines you will be using (other than the command to compile the current file) have been defined.

Using Projects and Project Spaces

Project spaces organize and house sets of projects. Once a project space is defined, it is primarily used to select or change projects and project spaces.

Selecting or Changing Projects

You can open or change a project in one of the following ways:

- Choose a project from the **Set Current** submenu of the **Project** menu. The **Set Current** menu lists all the projects contained by the current project space. The active project will have a check mark next to it. You may select from this list to reload any of the projects listed. Choosing a project in the **Set Current** menu is the same as opening the project.
- Choose a project from the list at the bottom of the **Project** menu.
- Choose **Project -> Project Space -> Open**, choose the filter `.PJT`, and navigate the directory structure for the project file. When you select the project file, a project space is opened as well, even if the project was not previously associated with a project space.

Selecting or Changing Project Spaces

You can select a project space by choosing **Open** from the **Project -> Project Space** submenu. Navigate your directory structure as you would when using the **File -> Open** dialog and select the project space (**.PSP**) file you wish to load.

- Recently selected project spaces will be listed at the bottom of the **Project Space** submenu, numbered from one to nine. You may select from this list to reload any of the project spaces listed.

Loading Files for Editing

One of the best things about projects is the convenience it provides when loading files. Just choose **Load Files** from the **Project** menu to choose from a list of the files in the current project.

- If you find that some files aren't listed, select the **Members** tab of **Project -> Settings** to add them.
- Select **Open** from the **File** menu to load files that are not part of the project. The rules for extended selection list boxes apply to the **Load Files** dialog.

Alternately, you may wish to use the **Project Window** to operate on files in a project. As with the **Load Files** dialog, you just select the desired files from the list and press **Enter** to load the files. The **Project Window** has the advantage of showing which files are actually on disk (some may currently be archived), and whether they are read/write (see the topic on [Characteristics of the File View Tab of the Project Window](#), in this chapter). In addition, you may define filters (file specification patterns) to sort the file list into groups (see the topic [The Filters Tab](#) in this chapter).

Creating, Selecting and Saving Workspaces

One of the most powerful aspects of projects is the workspace. At any time while working on a project, you can select **Save Workspace** from the **Project** menu and create a new workspace. Just give your workspace a name — a descriptive name this time, rather than just a file name — and you are done.



You must select a project from the **Set Current** submenu on the project menu before the workspace items on the **Project** menu will be available for use.

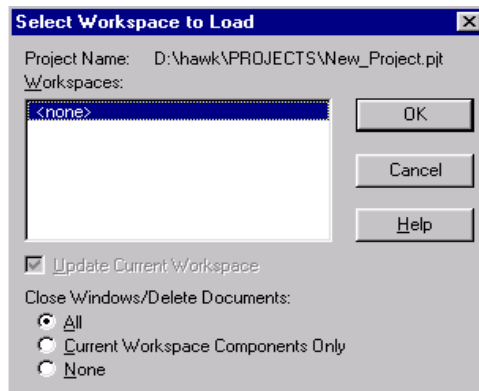
A workspace name may contain any printable characters (including spaces), but no apostrophe (**'**), backslash (****), quote (**"**), or square brackets (**[]**).

Creating a New Workspace

Suppose that you have a number of files open, and want to create a new workspace that does not include any of the files currently open. Complete the following steps:

1. Select the project that the workspace should be saved to using the **Set Current** submenu on the **Project** menu, and then select **Load Workspace** from the **Project** menu. The **Select Workspace to Load** dialog displays; this dialog is shown in [Figure 7-18](#).

Figure 7-18. Select Workspace to Load Dialog



2. You will see a workspace that is always available called `<none>`. Select that workspace to wipe the slate clean.
3. Set the `Close Windows/Delete Documents` option to `All` to close all your buffers and windows and start afresh, and click `OK`. Of course, if any of the buffers contain changes that have not been saved to disk you will be given an opportunity to do so.

Now that you have your clean slate, open the files that you want to have open in your workspace. At any point, you can save your workspace under whatever descriptive name you wish, by clicking `Save Workspace` on the `Project` menu.

Remember that workspaces are specific to the projects in which they are created.

Automatic Saving

Selecting `Save Workspace` from the `Project` menu is usually only necessary when creating a new workspace. Hawk automatically updates the active workspace whenever you:

- Close or change projects.
- Exit Hawk.
- (Optionally) when you change workspaces.

Loading an Existing Workspace

Once you have created more than one workspace, you will find it easy to switch between them. Select `Project -> Load Workspace`. You will notice a couple of options are available, as described below.

Update Current Workspace

When the `Update Current Workspace` box is checked, the workspace you are leaving is automatically updated before the new workspace is loaded.

Close Windows/Delete Buffers

Choose one of the following options when closing a workspace, with respect to the next workspace. Keep in mind that the saving of windows and buffers in the workspace you are leaving is unaffected by your selection.

- Choose **All** to have all files carry over into the next workspace. If you do this, we recommend that you either delete the carry-over buffers before saving the workspace, or save the workspace under a new name. Otherwise, workspaces will become less distinct, and therefore less useful, entities.
- Choose **Current Workspace Components Only** to close only the files that are members of the workspace. This means that any files you opened during the session, such as a special header file, will carry over to the next workspace.
- Choose **None** to close all windows and buffers after leaving one workspace and before the next is loaded. In this event, no buffers or windows are carried over from one workspace to the next.

Searching Project Files

You can search the entire group of files that are members of the current project space or project, or you can select a subset of the member projects or files on which to perform a search. This feature is available in the `Search and Replace Multiple Sources` dialog and is described in [Chapter 10 Search and Replace and Other Navigational Tools](#).

Selecting files for Check-in or Check-out

Hawk's version control interface (`Check-in`, `Check-out` and related commands) can operate independently of, or in conjunction with projects. The `Tools -> Version Control -> Maintenance` dialog allows you to select from files in the project or the current directory.

Configuration and State Hierarchy

When you have no projects in use, Hawk stores its configuration information in its configuration file (`MWHAWK.INI`). Between sessions, Hawk stores the more transient information about the files you are working on in its state file. When you are using a project, however, some configuration information is kept in the project file, and almost all of the state information is stored there also. The state file then serves largely to identify which project file you were using last.

You determine the amount of configuration information kept in the project file by selecting either the `<Default Settings>` item or a project from the `Project Properties` list box, and then setting options in the various tabs of the `Project -> Settings` dialog. The `Directories` tab in the dialog also has an option that allows additional items to be stored in the project file.

Choosing to set configurations as either `<Default Settings>` or project specific settings gives you the option of storing as much or as little information with projects as you like, and allows each project to have its own, unique configurations. These

configurations change as projects change. Also, storing more information in the project file means storing less information in the Configuration file, thereby minimizing the time it takes to load Hawk.

If you are using one or more workspaces within your project, document and window state information is stored with the active workspace section of the project file when you exit, rather than in the state section of the project. For more information about Hawk's main configuration files, see [Chapter 17 Configuration Files and Command Line Parameters](#).

8

The Hawk Debugger



This chapter describes using the Hawk Debugger. It includes the following sections:

[Overview](#)

[The Hawk Debugger Interface](#)

[Debugging User-State Applications](#)

[Debugging System-State Routines](#)

[Debugging User-State Applications Using LLROM](#)

[Installing the Debug Extensions Module for 68K Systems](#)

Overview

The Hawk Debugger is a Windows-based, graphical user interface for debugging OS-9 projects. It is based upon and uses most of the commands from the legacy OS-9 resident source-level debugger, `SrcDbg`.

Depending on which OS-9 modules you have loaded on your development system, you can use the Hawk Debugger to debug your OS-9 user-state applications and system-state code.

Starting the Hawk Debugger

There are two versions of the Hawk Debugger. One version is integrated into the Hawk IDE and is started by selecting **Debug** -> **Connect** from the Hawk main window. The other version is stand-alone and can be started from a command line by running the following program:

```
\MWOS\DOS\BIN\hawkdbgr.exe
```

The stand-alone Hawk Debugger can also be invoked automatically when a child process or thread is forked (see [Figure 8-4](#)). The stand-alone Hawk Debugger can be used for applications developed outside of the Hawk IDE. More than one instance of the stand-alone Hawk Debugger can be run simultaneously..



To correctly use the Hawk Debugger, your target system *must* have the `pipe` and `pipeman` modules in memory and available for use.

User-State Applications

User-state is the normal program environment in which processes are typically executed. Generally, user-state processes do not deal directly with the specific hardware configuration of the system. Instead, these applications run under the control of the OS-9 kernel, which performs the real-time task switching and scheduling on the processes.

OS-9 includes security features inherent in its process-model architecture. This architecture has the following security features:

- A user-state application runs without interfering with any other running process.
- A user-state application process does not interfere with the operating system code. Errant pointers or bad logic do not cause a system crash or cause other processes to fail.

System-State Routines

The OS-9 operating system creates the user-state environment and controls the processes running under its control. The OS itself, along with device drivers and file managers, operates in a different environment called system-state. In this environment, the code that runs has complete access to the system. Code can access memory at any location, and can cause processes running under OS control to fail.

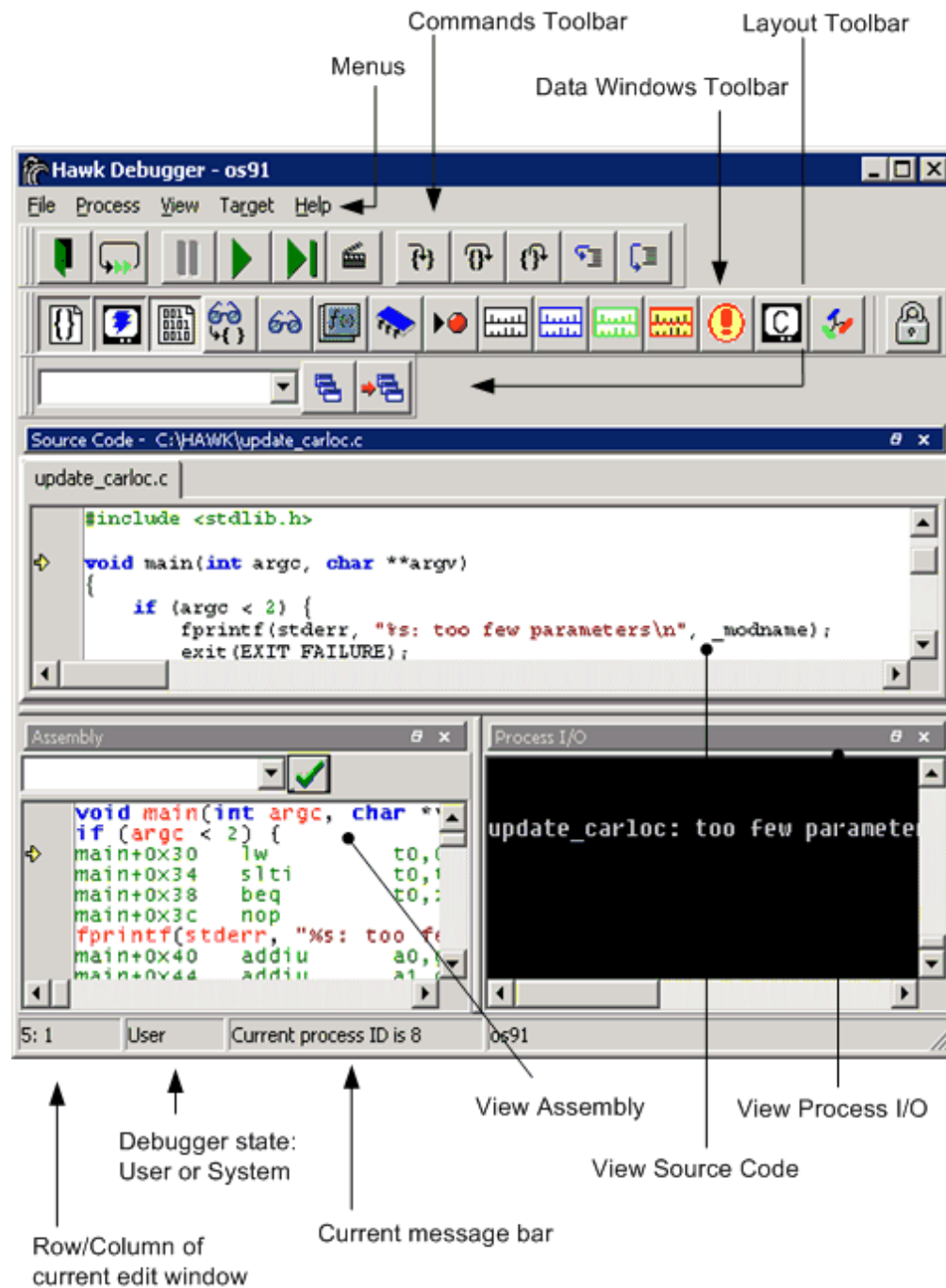
Since these applications are not under the control of the kernel, however, they do not follow the same restrictions or controls the kernel places on processes. This allows, for example, the efficient processing of interrupts by drivers.

Operating system extension modules (sometimes call P2 modules), trap handler modules and system-state alarms also operate in system-state. A process executes in system-state if the supervisor state bit in the module attribute/revision word is set and if the module is owned by the super user.

The Hawk Debugger Interface

The Hawk Debugger provides a graphical user interface for the OS-9 source-level debugger, `srcDbg`. This section documents the menu and toolbar functions. [Figure 8-1](#) shows the main Hawk Debugger window and some of the main components.

Figure 8-1. The Hawk Debugger



Hawk Debugger Functionality

The Hawk Debugger functionality is accessed through the menus and tool bars. Some of the functionality can be accessed through both means.

File Functions

From the File menu you open and print source files, set the Hawk Debugger user options, and save and select your user-defined layout configurations. You also exit the Hawk Debugger from the File menu. Figure 8-2 shows the File menu.

Figure 8-2. File Menu

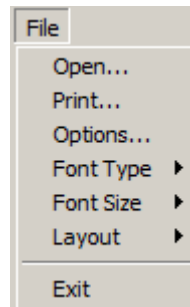
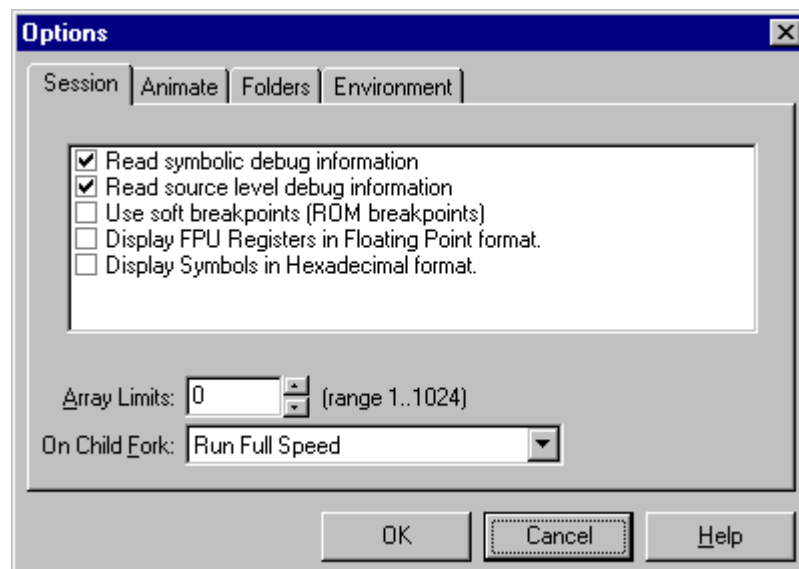


Figure 8-3 through Figure 8-7 are associated with the Hawk Debugger user Options, which are located in the Debug file menu or the Options button on the Debugger Connect window.

The Session tab, shown in Figure 8-3, enables you to set the following options:

Figure 8-3. Session Tab



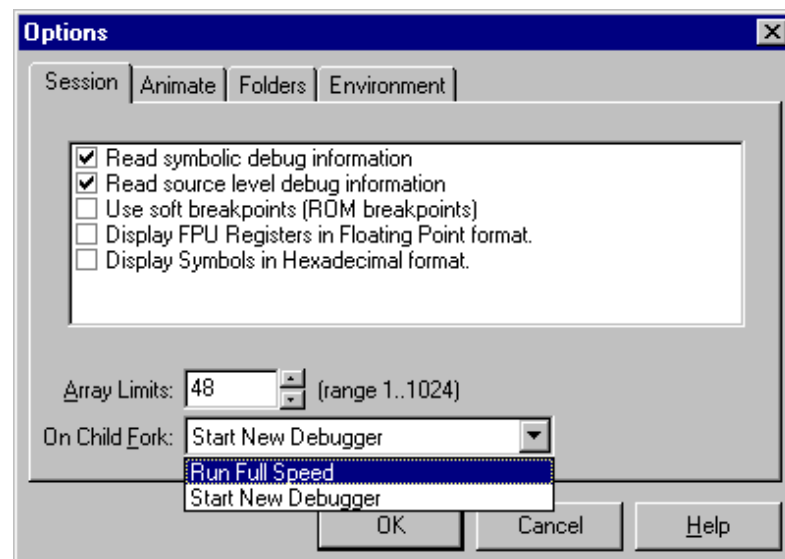
- Read symbolic debug information causes the Hawk Debugger to process the .stb file when loading or attaching to modules. The .stb file contains basic symbolic information for global variables and functions.
- Read source level debug information causes the Hawk Debugger to process the .dbg file when loading or attaching to modules. The .dbg file contains source level debugging information.
- Use soft breakpoints is important because soft breakpoints are typically used when the code being debugged is stored in read-only memory (ROM.)

Instead of placing a special processor instruction at the breakpoint location, here the program counter is compared with the list of breakpoints after the execution of each instruction. Because of this, programs being debugged with soft breakpoints execute slower than when normal “hard” breakpoints are used.

- `Display FPU Registers in Floating Point format` causes the Hawk Debugger to display the contents of Floating Point Unit (FPU) registers as decimal fractions. When not selected, the contents of FPU registers are shown as a hexadecimal number.
- `Display Symbols in Hexadecimal format` causes the value of symbol to display in hexadecimal format. When this option is not selected, symbol information is displayed in decimal format.

The Sessions tab also includes the `On Child Fork` menu. The options are shown in [Figure 8-4](#), and include the following:

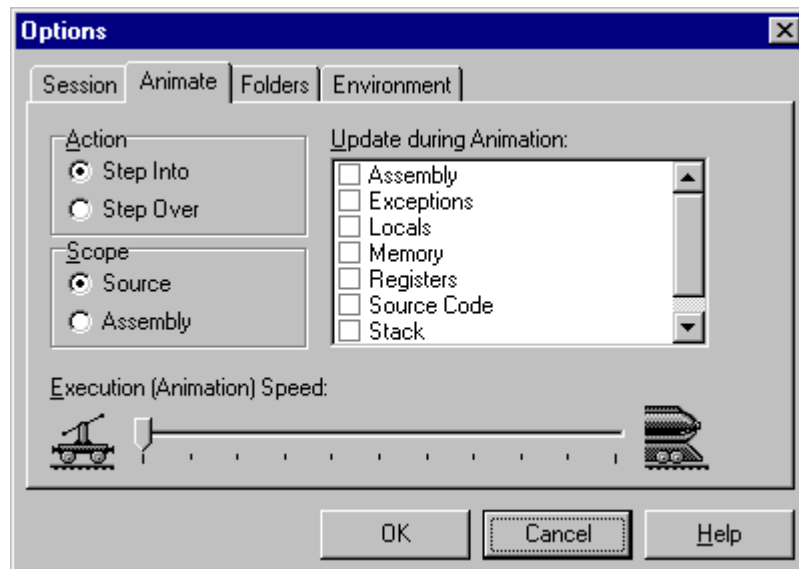
Figure 8-4. On Child Fork Menu



- `Run Full Speed` causes processes forked by the program being debugged to run normally.
- `Start New Debugger` causes a new Hawk Debugger window to display the forked process.

[Figure 8-5](#) shows the Animate tab options. These include the following:

Figure 8-5. Animate Tab



- **Action**
 - `Step Into` causes the Hawk Debugger to step into function calls.
 - `Step Over` causes the debugger to treat function calls as one machine instruction.
- **Scope**
 - `Source` causes the Hawk Debugger to step through the executing program one C source line at a time.
 - `Assembly` causes the debugger to step through one machine instruction at a time.
- **Update during Animation**
 - Selecting a check box causes the Hawk Debugger to update the selected window during the animation process. For example, if the `Registers` check box is selected and the `Registers` window is open, the debugger will update the register display after each step through the program.
 - Updating the windows requires more data to be sent from the target system. Therefore, as you enable more updating, the animation process becomes slower.
- **Execution (Animation) Speed**
 - This slider sets the duration between steps through the program. The handcart represents the slowest setting and the locomotive represents the fastest setting.

Figure 8-6 shows the `Folders` tab options. This enables you to set the search paths for the desired source and object code. The symbolic information files (`.stb` and `.dbg`) are searched for in the **Object Code** directories.

Figure 8-6. Folders Tab

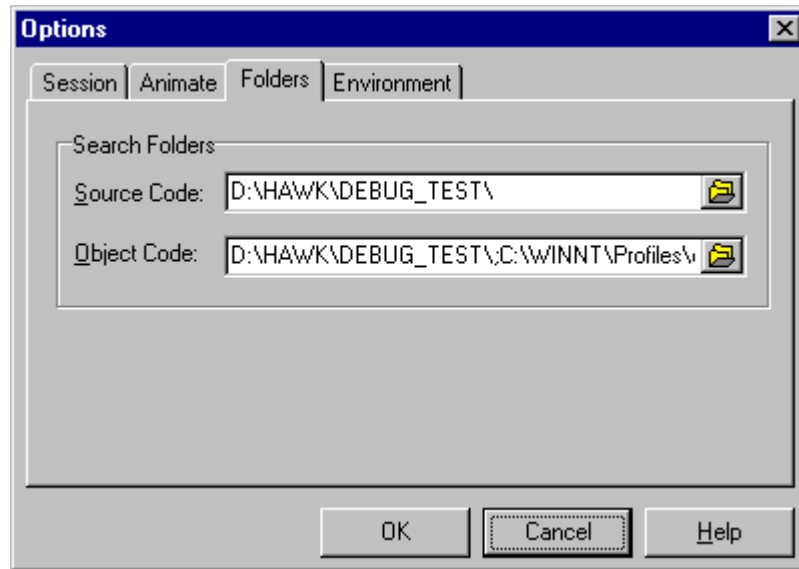
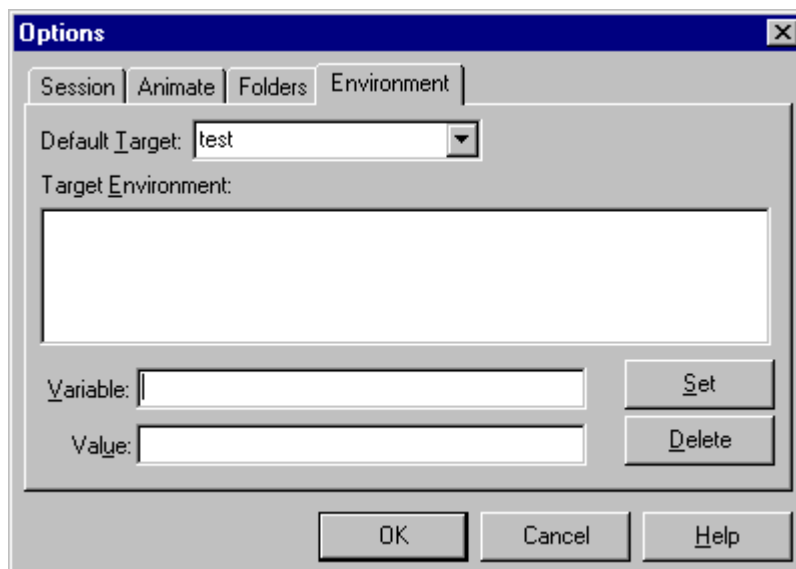


Figure 8-7 shows the Environment tab options. This tab is used to set environment variables for the debugged process. When the process is forked, these environment variables will be passed to the target program's main function. The Default Target pulldown list specifies the default target system to be used for debugging.

In addition, when a process being debugged forks a new thread, it is treated in a manner similar to the forking of a new process. The Debugger can either start a new debug session for the new thread or optionally allow the new thread to run without debug control.

Figure 8-7. Environment Tab



The File menu also includes a Layout selection, shown in Figure 8-8. This functionality is also accessible through the Layout Toolbar, also shown in Figure 8-8. Once the user

interface is configured as desired, the configuration can be named, saved, and used repeatedly. You can also save more than one configuration.

Figure 8-8. Layout Options



Commands

The Hawk Debugger commands, the actual debugging operations performed on the project's code, are accessible through the `Process` menu and the `Commands` toolbar. These are shown in [Figure 8-9](#) and [Figure 8-10](#) respectively. Each command function is described briefly in [Table 8-1](#).

Figure 8-9. Process Commands Menu

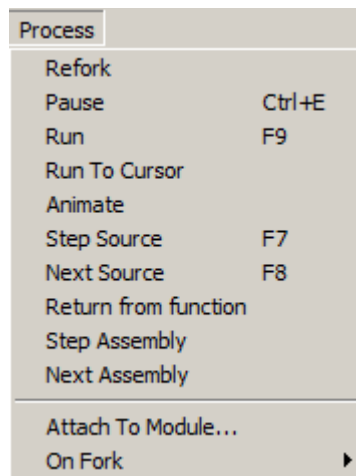


Figure 8-10. Process Commands Toolbar

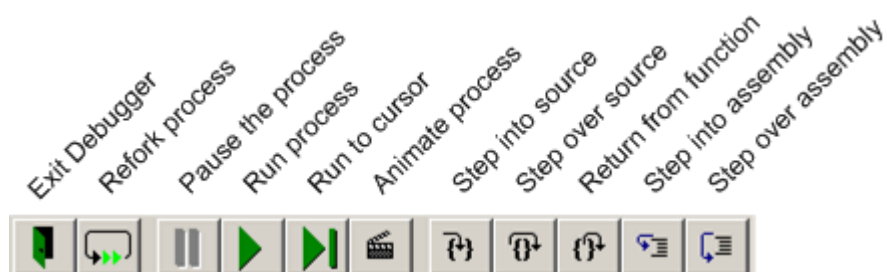


Table 8-1. Hawk Debugger Commands

Command	Description
Exit Debugger	Terminates your current debug session.
Refork process	Causes the current debug process to be exited and the program to be re-forked.
Pause the process	Stops an executing process.
Run process	Starts a stopped process. Execution resumes from the current program counter value.
Run to cursor	The process is run up to the source line indicated by the cursor.
Animate process	The current process is stepped through automatically. The rate of execution and which windows are updated is controlled by the Animate tab in the Option dialog.
Step into source	Steps one C/C++ source line. If a function call is encountered, the debugger steps into the function.
Step over source	Steps one C/C++ source line. If a function call is encountered, the debugger steps over the call.
Return from function	Causes the debugger to run the target process until it returns from the currently active function.
Step into assembly	Steps one machine instruction (assembly source line) at a time. If a subroutine call is encountered, the debugger steps into the subroutine.
Step over assembly	Steps one machine instruction (assembly source line) at a time. If a subroutine call is encountered, the debugger treats the subroutine call as a single instruction.

In addition, the Process menu includes an `Attach to Module` selection. This selection enables you to attach to a library or program module on the target system. The Hawk Debugger will attempt to locate and process the associated `.stb` and `.dbg` files as enabled by the `Session` tab in the `Options` dialog (see [Figure 8-3](#) for details).

View Options

The Hawk Debugger view options determine how the Hawk Debugger interface appears as you are working. These options are controlled through the View menu and the Data Windows toolbar. These are shown in [Figure 8-11](#) and [Figure 8-12](#) respectively. Each command function is described briefly in [Table 8-2](#).

Figure 8-11. View Menu



Figure 8-12. Datawindows Toolbar



Table 8-2. Hawk Debugger View Options

View Option	Description
Toggle Source code visibility	Toggles the display of the C/C++ source window.
Toggle Process I/O visibility	Toggles the display of the Target Process I/O window.

Table 8-2. Hawk Debugger View Options (Continued)

View Option	Description
Toggle Assembly visibility	Toggles the display of the Assembly Source Code Window.
Toggle Locals visibility	Toggles the display of the Local Variables Window.
Toggle Watch visibility	Toggles the display of the Watch Expression window.
Toggle Call Stack visibility	Toggles the display of the Call Stack window.
Toggle Memory visibility	Toggles the display of the Target Memory window.
Toggle Breakpoints visibility	Toggles the display of the Breakpoint window.
Toggle MPU Registers visibility	Toggles the display of the MPU Registers.
Toggle FPU Registers visibility	Toggles the display of the FPU Registers.
Toggle MMU Registers visibility	Toggles the display of the MMU Registers. These registers are only viewable when debugging in system state.
Toggle "Special" Registers visibility	Toggles the display of special processor control registers. These registers are only viewable when debugging in system state.
Toggle Exceptions visibility	Toggles the display of the Exceptions window. This window is only active when debugging in system state.
Toggle Command visibility	Toggles the Command I/O window. The Command I/O window allows you to communicate via command-line commands to the debugger.
Toggle Symbol Browser visibility	Toggles the Symbol Browser window. This window can be used to enable/disable breakpoints. It is especially helpful when multiple modules' symbols are available.
Lock the docked windows	Prevents docked windows from becoming undocked.

The View menu and Data Windows toolbar also enable control of the registers displaying in the interface. Figure 8-13 shows these control points. As with other view options, the register windows can be docked and undocked, and placed on the screen as desired. Figure 8-14 shows a Hawk Debugger interface displaying registers.

Figure 8-13. Register View Options

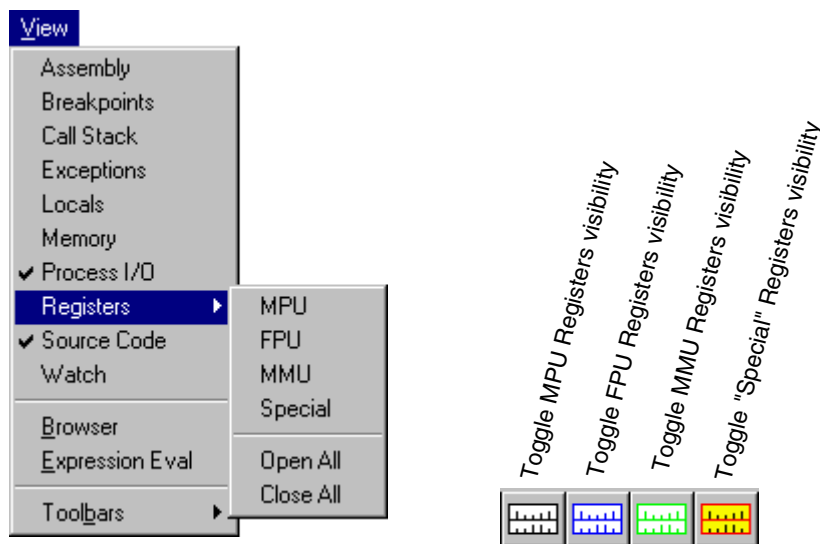
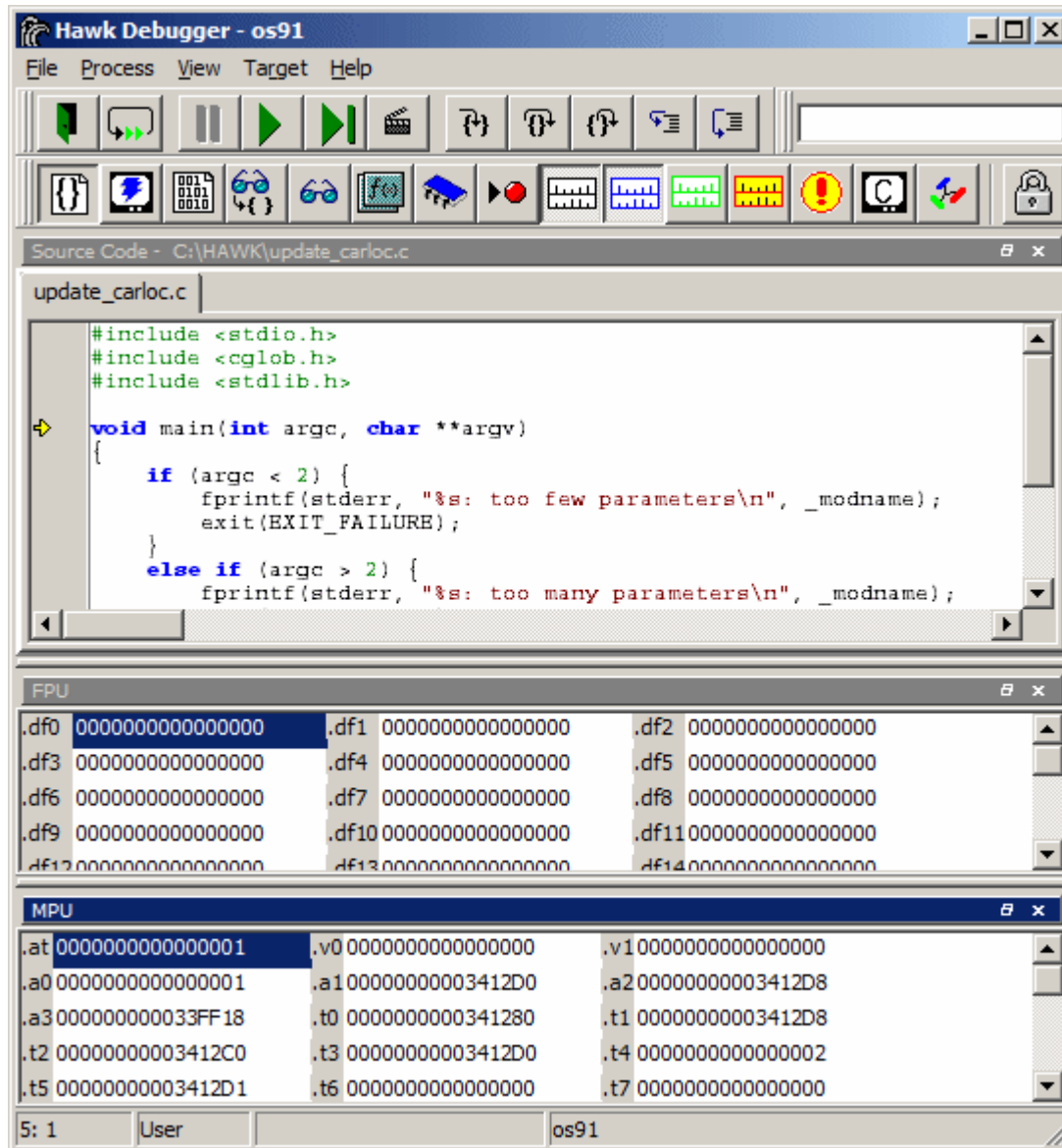


Figure 8-14. Registers Display



The View menu also controls which toolbars display in the interface. [Figure 8-15](#) shows this control. [Figure 8-16](#) shows the Hawk Debugger toolbars. All toolbars can be docked and undocked within the interface window. The Commands toolbar can be undocked and placed outside of the interface window.

Figure 8-15. Toolbar View Options

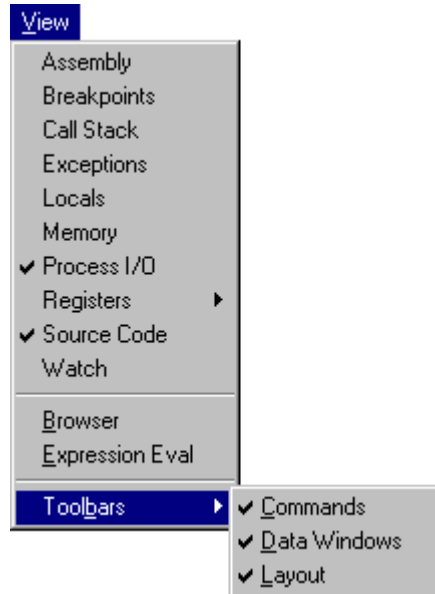
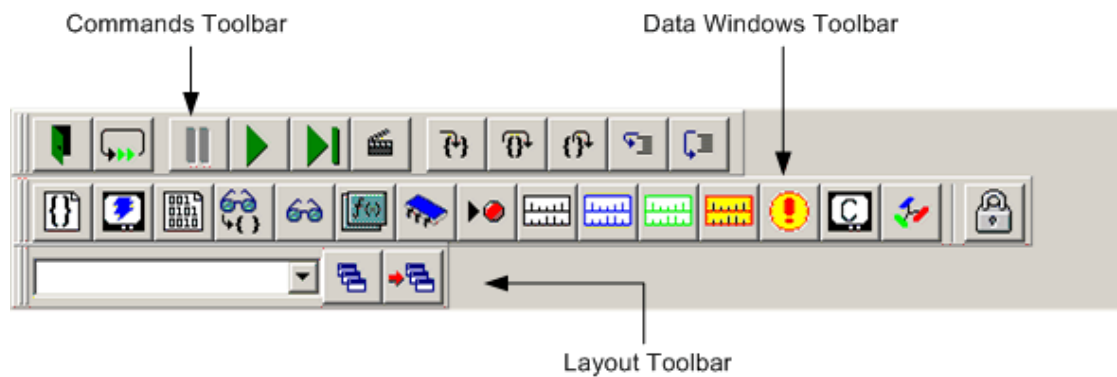


Figure 8-16. Hawk Debugger Toolbars



Browser Tool

The Browser tool is provided under the View menu. Selecting the `Browser` option, shown in [Figure 8-17](#), displays the `Symbol Browser`. You can use the Symbol Browser to examine the block structure of the program being debugged and to set breakpoints, set context, evaluate an expression or add a variable to the Watch window.

The Browser toolbar, also shown in [Figure 8-17](#), is described in [Table 8-3](#).



If you are having trouble setting breakpoints in Hawk, use the `Set Context` feature. This feature is located in Hawk's Symbol Browser and is defined in [Table 8-3](#). Steps on how to use the feature follow:

To use the `Set Context` feature, complete the following steps:

1. Open the Symbol Browser (`Debug -> View -> Browse Symbol`) and highlight the desired function.
2. Right-click on the function and select the `Set Context` option. The corresponding source file is opened in the Editor Window and a black arrow outlined in yellow identifies the new context location.

You should now be able to toggle breakpoints from within the Editor Window by highlighting the desired line, right clicking, and selecting the `Toggle Breakpoint` option. (You can then also toggle breakpoints by left click in the border to the left of the desired line.)

Figure 8-17. Browser Window

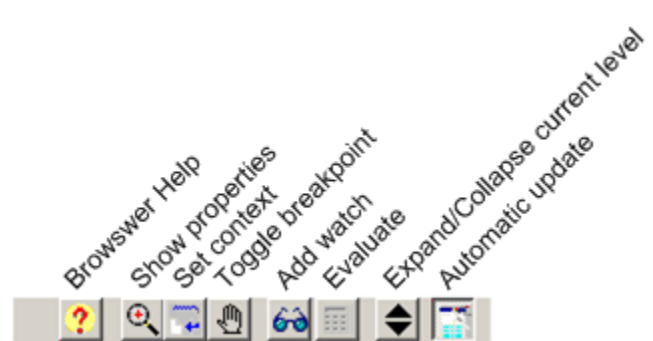
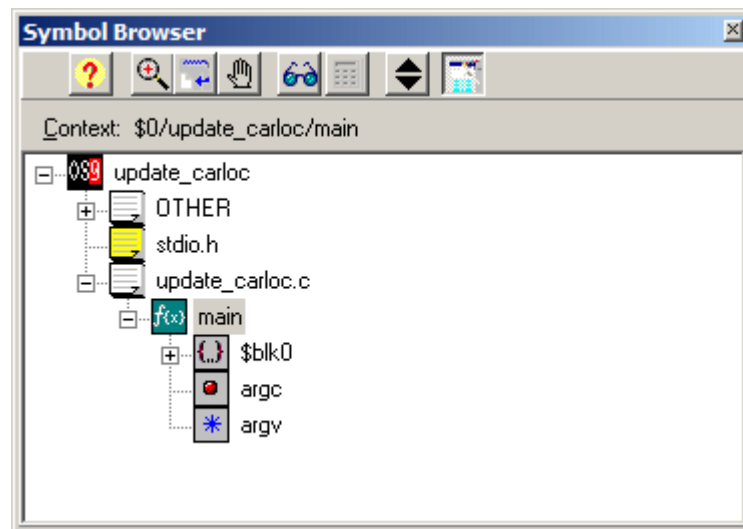
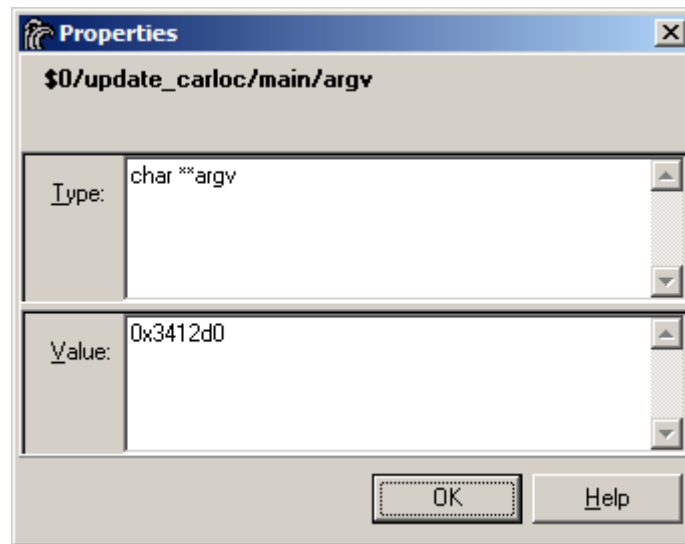


Table 8-3. Browser Toolbar Options

Tool	Description
Browser Help	Brings up the help information for the Browser.
Show properties	This button causes a properties window to be displayed as shown in Figure 8-18 . The type and value of the selected item is displayed.
Set context	Causes the debugger's context to be set to the selected function or block. If there is an active stack frame for the selected function or block, the variables that are in scope can be evaluated.
Toggle breakpoint	Toggles a breakpoint at the selected function or block.
Add watch	Adds the selected data item to the Watch window.
Evaluate	Causes the selected data item to be evaluated and the results displayed in an Expression Evaluation window.
Expand/Collapse current level	Toggles the view of the element list.
Automatic update	Automatically updates the view according to a preset time interval.

Selecting the `Show Properties` button on the Browser toolbar, displays the properties window shown in [Figure 8-18](#). The type and value of the selected item is displayed.

Figure 8-18. Show Properties Window



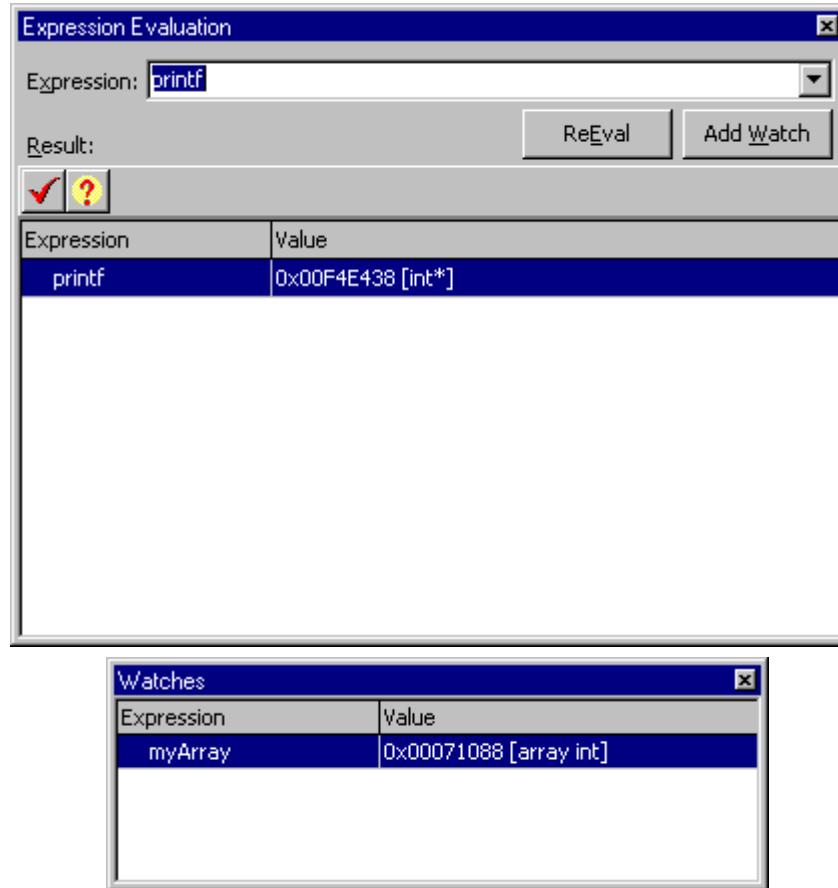
Expression Eval Tool

The Expression Eval tool, shown in [Figure 8-19](#), displays the Expression Evaluation window. This shows the current value of a variable or expression.

The Add Watch button in the Expression Evaluation window, also shown in [Figure 8-18](#), displays the value of a C/C++ expression.

Once opened, the Watches window can be docked and undocked within the interface.

Figure 8-19. Expression Evaluation Window



Debugging User-State Applications

Every OS-9 kernel has debugging support built into it, which allows the kernel to actually control the process that is being debugged. Since this support is built-in to the kernel, it is rather a simple process to debug an OS-9 process. Debugging is implemented as a client-server methodology, where the Windows host machine is the client and the OS-9 machine is the server.

Figure 8-20 shows the major components involved in a debug session as well as the communications links between them. The four components include the following:

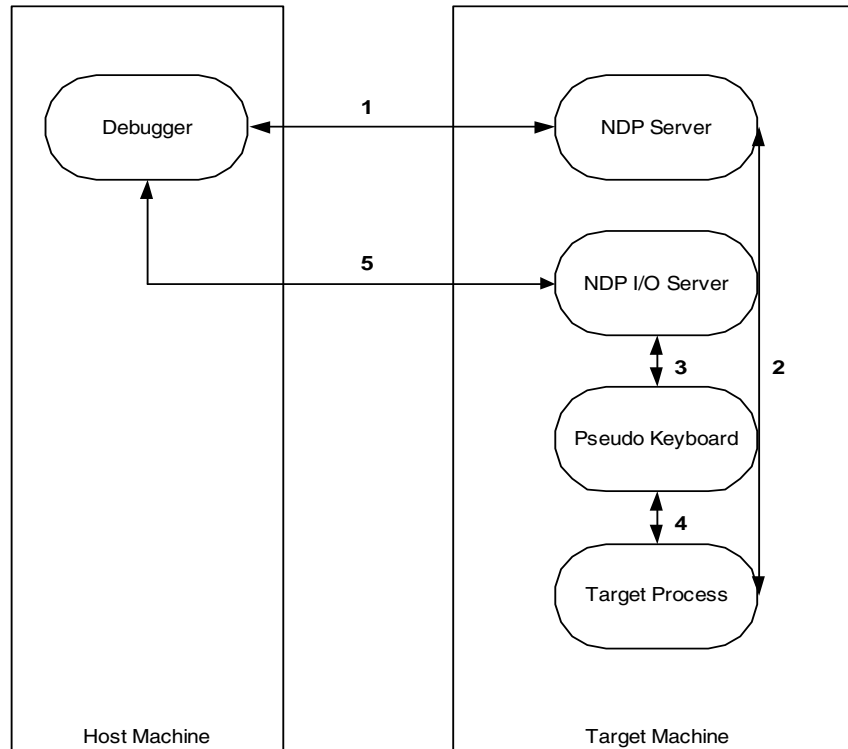
- debugger running on Windows
- NDP Server (*spfnndpdc*)
- NDP I/O Server (*ndpio*)
- Target Process running on OS-9.



Using the Hawk Debugger requires that the *pipe* and *pipeman* modules be loaded in the target system's memory.

There are five bi-directional connections between the different components. The following sections describe the components and the connections between them.

Figure 8-20. Debugging Components and Connections



NDP Server Functions

The NDP Server (`spfnpd.c`) controls execution of a Target Process on behalf of the debugger. Commands and replies are passed back and forth using TCP/IP. As well as controlling the Target Process, the NDP Server is responsible for setting up the I/O channel between the debugger and the Target Process's standard I/O paths (only paths 0, 1 and 2 are monitored).

Control of the Target Process is accomplished using specific debugging system calls provided by OS-9. In response to requests from the debugger, the NDP Server manipulates target memory and target registers, sets breakpoints and executes the program. Because of the way the debugging system calls work, the NDP Server is inactive whenever the Target Process is executing. This has the side effect that the debugger cannot communicate with the NDP Server while the Target Process is executing. Because of this, a separate process is needed to allow standard input and output facilities between the Target Process and the debugger.

NDP I/O Server

The NDP I/O Server redirects I/O from the debugger to the Target Process and vice versa. The data is transferred to and from the Target Process by way of the Pseudo Keyboard Manager. This gives the effect of the Target Process talking to a terminal. A separate process is required for this task because, as discussed above, when the Target Process is running, the NDP Server is inactive and cannot be involved in transferring I/O data.

Connections

Debugger/NDP Server Connection

The connection between the debugger and the NDP Server (1) is a TCP connection carrying NDP packets bi-directionally. This connection is established by the Network Debugger Protocol Daemon (`spfnbdp`). The port number for this connection is 13312 (0x3400).

The NDP protocol is based on the LDP protocol described in RFC-909. Several changes have been made to improve throughput.

NDP Server/Target Process Connection

The connection between the NDP Server (2) and the Target Process is by way of the OS-9 kernel's `_os_dfork`, `_os_dexec`, and `_os_attach` system calls.

NDP I/O Server/Pseudo Keyboard Connection

The connection between the NDP I/O Server and the Pseudo Keyboard Manager master port (3) carries input and output to and from the Target Process. Data from this connection is passed onto the debugger through connection number (5) shown in [Figure 8-20](#). Data coming in from connection number (5) is given to the Pseudo Keyboard Manager. This connection is established by the NDP Server, which forks the NDP I/O server.

Target Process/Pseudo Keyboard Connection

The connection between the Target Process and the Pseudo Keyboard Manager slave port (4) provides the standard input and output paths for the Target Process. Data from the Target Process is passed through this connection to the Pseudo Keyboard Manager and from there, to the NDP I/O Server.

NDP I/O Server/Debugger Connection

The connection between the debugger on the Host Machine and the NDP I/O Server on the Target Machine (5) carries the standard input and output to and from the Target Process. This connection is set up by cooperation between the debugger and the NDP Server. The connection is a bi-directional TCP path.

Necessary Software

To debug using this methodology requires a stable communications package on the host and target using TCP/IP. The supported Windows platforms have this available as a default and it is quite reliable.

Required Modules

For user-state debugging, you need the following modules loaded on your target system:

- `spfndpd`
- `spfndpdc`
- `ndpio`
- `pkman`
- `pkdrv`
- `pk`
- `pks` (68k only).



The list above assumes that the appropriate underlying network software, including SoftStax® and the LAN Communications with support for Ethernet, SLIP or PPP, is in place on the OS-9 target machine.



For an example of user-state debugging, see the *Getting Started with Hawk* manual.

Debugging System-State Routines

Debugging in system-state with OS-9 is a bit more complex than debugging in user-state, but the basic client-server model is still the same. However, an additional module is needed to control the system while debugging, since the debugging services in the kernel are no longer available. This module, called `dbgserver` (or the Debug Server) provides similar debugging services as the kernel does.

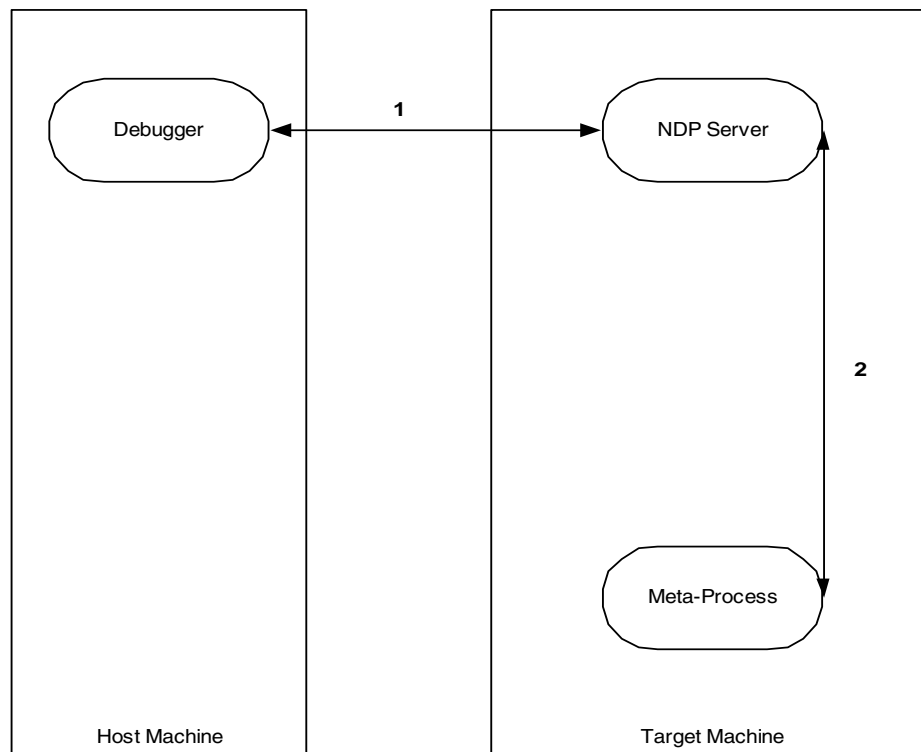
The other main difference between system- and user-state debugging is what is actually being debugged. In user-state debugging, you are debugging one single OS-9 process. In system-state debugging, you are debugging the entire target image as one monolithic entity. This means you are debugging the OS, drivers, file managers, and all the running processes as one meta-process. Debugging at this level has some interesting results. For example, when you stop at a breakpoint while system state debugging, the entire meta-process comes to a halt. This means that all processes stop executing, no interrupts are processed, etc. After resuming from a breakpoint, the meta-process resumes and everything starts running again.

[Figure 8-21](#) shows the major components involved in a system-state debug session as well as the communications links between them. The three components include the following:

- debugger running on Windows
- NDP Server (`sndp`)
- Meta-Process running on the target

The following sections describe the components and the connections between them in some detail.

Figure 8-21. Components of System-state Debugging Session



NDP Server Functions

The NDP Server (`sndp`) is a system-level module that controls execution of the Meta-Process on behalf of the debugger. Commands and replies are passed back and forth using TCP/IP.

Control of the Target Process is accomplished using specific debugging calls provided by the Debugger Server, `dbgsevr`. In response to requests from the debugger, the NDP Server manipulates target memory and target registers, sets breakpoints and executes the program.

Connections

Debugger/NDP Server Connection

The connection between the debugger and the NDP Server (1) is a TCP connection carrying NDP packets bi-directionally. This connection is established by the NDP Server (`sndp`). The port number for this connection is 13312 (0x3400).

The NDP protocol is based on the LDP protocol described in RFC-909. Several changes have been made to improve throughput.

NDP Server/Meta Process Connection

The connection between the NDP Server (2) and the Meta-Process is established using the Debug Server (`dbgserver`) API calls.

Restrictions

One of the major restrictions of the system-state client-server configuration is the handling of system-state I/O. Due to restrictions in the communication software used for system-state debugging, there is no ability to handle I/O generated by the Meta-Process during debugging. Therefore, any I/O generated is displayed on the designated OS-9 console, if one exists.

Communications Layer for System-state Debugging

Using the communications layer for system-state debugging requires a stable communications package on the host and target using TCP/IP. The supported Windows platforms have this available as a default and it is quite reliable.

The OS-9 communications layer is a bit different in system-state debugging than in user-state. Since we are debugging the Meta-Process and several pieces of the Microware LAN Communications run as processes under OS-9, system-state debugging cannot be performed using this communications package. Instead, another communications package needs to be used called Low-Level ROM Communications, or LLROM.

This communications package provides minimal TCP/IP services without the use of interrupts or other support that the OS-9 kernel provides. Instead, it uses a polled architecture, which does affect performance, but provides the support needed in a system-state debugging environment. LLROM is configured when you configure the OS-9 environment.

Required Modules

For system-state debugging, you need the following modules loaded on your target system:

- `sndp`
- `dbgserver`
- `dbgentry`

- `break`.



This list assumes that the appropriate underlying networking is in place. For system-state debugging, the underlying network includes the low-level communications modules.



For an example of system-state debugging, see the *Getting Started with Hawk* manual.

Debugging User-State Applications Using LLROM

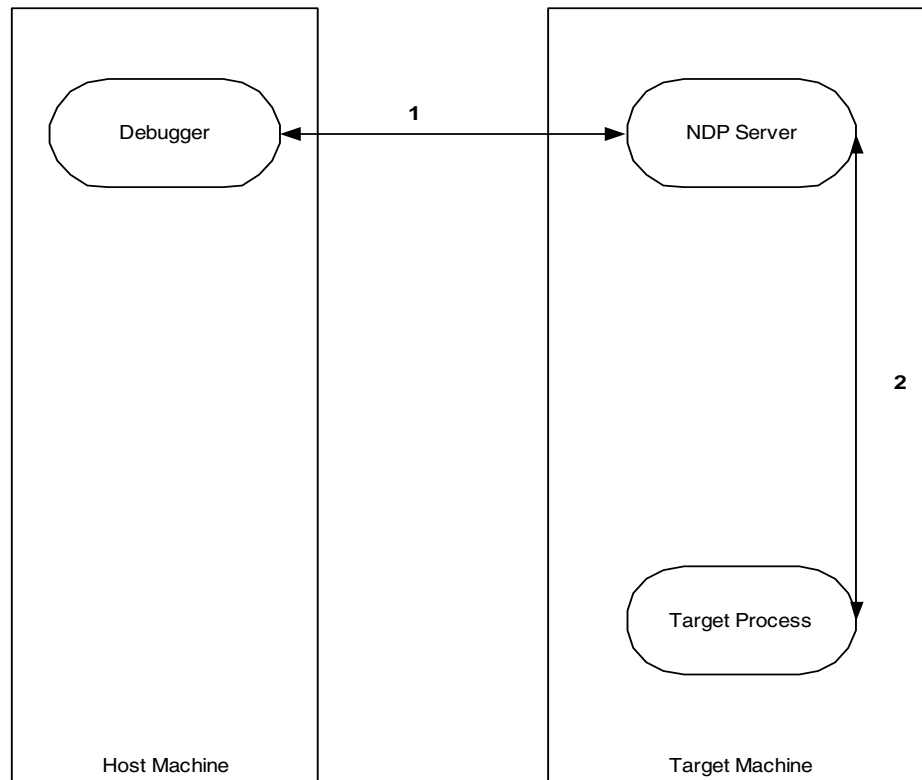
Many times, embedded hardware does not have resources to provide the full Microware LAN Communications implementation. In these cases, there is a third debugging solution on OS-9 that provides user-state debugging using the LLROM communications layer that is used for system-state debugging.

Figure 8-22 shows the major components involved in a debug session as well as the communications links between them. The three components are:

- debugger running on Windows
- NDP Server (`undpdc`)
- Target Process running on OS-9

The following sections describe the components and the connections established between them in some detail.

Figure 8-22. Debug Components and Communication Links



NDP Server Functions

The NDP Server (`undpdc`) controls execution of a Target Process on behalf of the debugger. Commands and replies are passed back and forth using TCP/IP.

Control of the Target Process is accomplished using specific debugging system calls provided by OS-9. In response to requests from the debugger, the NDP Server manipulates target memory and target registers, sets breakpoints and executes the program. Because of the way the debugging system calls work, the NDP Server is inactive whenever the Target Process is executing.

Connections

Debugger/NDP Server Connection

The connection between the debugger and the NDP Server (1) is a TCP connection carrying NDP packets bi-directionally. This connection is established by the Network Debugger Protocol Daemon (`undpd`). The port number for this connection is 13312 (0x3400).

The NDP protocol is based on the LDP protocol described in RFC-909. Several changes have been made to improve throughput.

NDP Server/Target Process Connection

The connection between the NDP Server (2) and the Target Process is established using the OS-9 kernel's `_os_dfork`, `_os_dexec` and `_os_attach` system calls.

Restrictions

Due to restrictions in the LLROM communication software used for debugging, there is no ability to handle I/O generated by the target process during debugging. Therefore, any I/O will be on the path(s) (`stdin`, `stdout`, `stderr`) on which `undpd` was started. If a process is forked from the Debugger, it will start with the path(s) specified at fork time, and I/O will be generated on those paths.

Necessary Software

To debug using this methodology requires a stable communications package on the host and target using TCP/IP. The supported Windows platforms have this available as a default and it is quite reliable.

Required Modules

For low-level user-state debugging, you need the following modules loaded on your target system:

- `undpd`
- `undpdc`
- `hlproto`



The list above assumes that the appropriate underlying network software, including the applicable low-level network I/O modules (also known as LLROM) with support for Ethernet or SLIP, is in place on the OS-9 target machine.

Installing the Debug Extensions Module for 68K Systems

The Debug Extensions module, `dbgextns`, extends the OS-9 for 68K kernel to enable new debugging features provided by the Hawk Integrated Development Environment. These include attaching to running processes and debugging children forked by processes being debugged.

To enable these features, `dbgextns` must be loaded in memory on the target system and must also be initialized. You can initialize `dbgextns` using one of the following methods:

1. Using the OS-9 `p2init` utility.
2. Including `dbgextns` in the OS-9 bootfile and including its name in the list of extensions in the `init` module.

The `dbgextns` module can be found in the following directories, depending on the type of host system:

- 68000,68010, CPU32, and 68070


MWOS/OS9/68000/CMDS/BOOTOBS

- 68020, 68030, 68040, and 68060

MWOS/OS9/68020/CMDS/BOOTOBS

9

Version Control



Hawk does not come with its own version control system, but can be configured to integrate with an existing system. This chapter describes the Hawk version control features. It includes the following sections:

[Using Version Control in Hawk](#)

[Version Control Integration Configuration](#)

Using Version Control in Hawk

This section describes the Hawk version control user interface. You must have an existing version control system integrated with Hawk using one of the two methods described in the [Version Control Integration Configuration](#) section. The appropriate integration method to use depends on the type of version control system available.

Version Control Menu

Most version control operations in Hawk can be carried out from the `Tools -> Version Control` submenu. From this menu, the following capabilities are available:

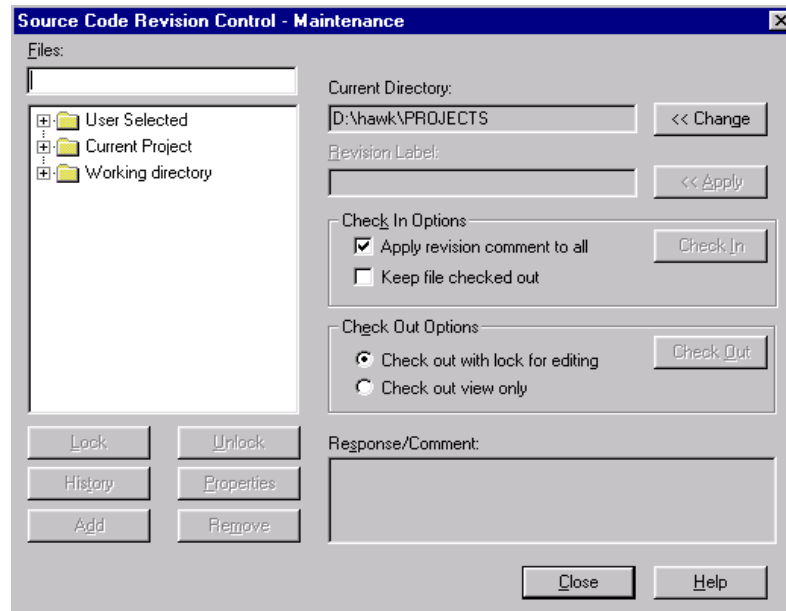
- Perform standard `put`, `get`, `get-with-lock`, `lock`, and `unlock` commands.
- See the properties and histories of any files under version control using the `Properties` and `History` items.
- Add or remove files from the version control project by clicking on `Add` or `Remove`.
- Access the Version Control Manager for the version control provider using the `Manager` item.

Source Code Revision Control—Maintenance Dialog

While most of the items on the `Version Control` submenu are self-explanatory, an item that requires deeper explanation is `Maintenance`, which accesses the `Source Code Revision Control Maintenance` dialog. This dialog is a multi-purpose dialog that works for either the command line interface or the SCC Provider interface.

This dialog is also available on the right-mouse-click pop-up menu in the `Hawk Project Window` (which is displayed by default on the left-hand side of the main Hawk window). If the `Project Window` doesn't appear in Hawk, click the `Project` menu item in the `Window` pull-down menu.

Figure 9-1. Source Code Revision Control Maintenance Dialog



The Maintenance dialog is useful for the following actions:

- applying labels to revisions
- locking and unlocking revisions without checking them in or out
- reviewing the revision history and properties of an archive
- performing other version control tasks

Some of these services, such as history and properties, are very specific to the version control system or SCC provider you are using.



Any file opened in Hawk that you wish to perform version control operations on must already be part of a version control project. In the case of the SCC interface, it must be a part of the version control project that you have opened under **Tools -> Version Control -> Setup**. The file does not have to be part of a Hawk project, although there are advantages to using Hawk projects in conjunction with existing version control projects.

Version Control and Hawk Projects

Refer to this section to complete the following:

- Associate Version Control projects with Hawk projects.
- Add version control project filenames to your Hawk project.
- Add Hawk project files to an SCC Provider Project.

Associating Version Control Projects with Hawk Projects

It is possible to associate version control projects with Hawk projects when using SCC API version control integration. This is described in the [Hawk SCC Integration with Version Control Systems](#) section. Doing so offers the following advantages:

- When a Hawk project is opened, the associated version control project is also automatically opened, thus making version control facilities readily available to be used from within Hawk. `Check-in` and `Check-out` (among other version control commands) are immediately available, allowing for quick preparation of files for editing.
- Closing the Hawk project automatically closes the associated version control project.
- Files in the associated version control project can be "read" into the Hawk project, providing a more convenient way to add appropriate files to the Hawk project.

A version control project can be associated with a Hawk project from one of two different dialogs: the `Tools -> Version Control -> Setup` dialog or the `Project -> Settings -> Members` dialog. To associate the two projects in the `Tools -> Version Control -> Setup` dialog, complete the following steps:

1. Create a new project space with at least one member project (or open an existing one).



For information about creating projects, see [Chapter 7 Projects, Project Spaces, and Workspaces](#).

2. Set a current Hawk project using the `Set Current` menu item on the `Project` menu.
3. Go to `Tools -> Version Control -> Setup`.
4. Choose the `SCC Provider` radio button.
5. Assuming that the version control provider has already been initialized, click `Open Project`.
6. Choose the version control project that you want to associate with the current Hawk project.
7. Verify that the working directory listed is that of the version control project's, not the Hawk project's (or any other) directory. If the directory is not the one that was specified for the version control project, make it so by browsing for, or typing in, the correct directory name.
8. Click `OK`, and then `OK` again.

To associate Hawk and version control projects in the `Project -> Settings -> Members` dialog, complete the following steps:

1. Assuming that a project space has been created with at least one member project, go to the `Project -> Settings -> Members` dialog.

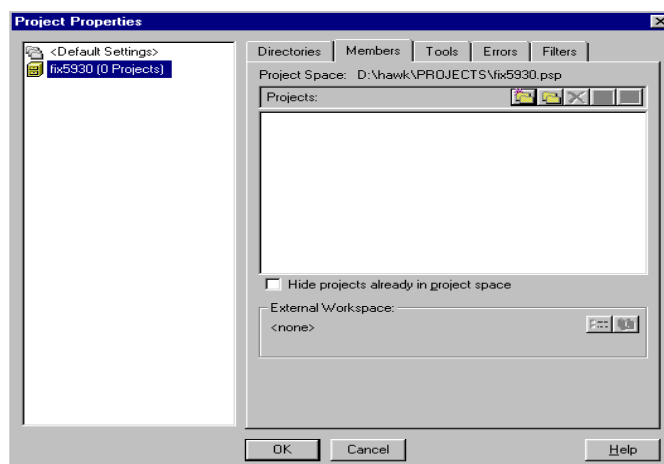
2. Highlight the Hawk project that you want to associate with the version control project.
3. In the `VCS Project` group, click the **Setup** button (the group will not be available if a version control provider has not been installed and initialized).
4. Choose the version control project that you want to associate.
5. Verify that the working directory listed is that of the version control project's, not the Hawk project's (or any other) directory. If the directory is not the one that was specified for the version control project, make it so by browsing for, or typing in, the correct directory name.
6. Click **OK**, and then click **OK** again.

Add Version Control Project Files to a Hawk Project

Filenames that make up version control projects can easily be added to Hawk projects when using the SCC service provider version control integration.

To add version control files to a Hawk project, use the VCS Project `Read` feature. This feature will read or scan a version control project or configuration file for files, in order to incorporate those files into a Hawk project. The version control "Read" feature can be found in the `Project -> Settings -> Members` dialog.

Figure 9-2. Project -> Settings -> Members Dialog



Address the following items that have to do with Hawk's version control "Read" feature:

- `VCS Project`: Lists the name of this version control project.
- `Setup` button: Allows you to associate an SCC Project, or change the association. Complete the dialog that displays.
- The `Read` button: Allows you to populate the project files list box.








Having Hawk automatically read the filenames of a version control project into a Hawk project saves the time needed to manually add files, one by one, to a Hawk project.

Add Hawk Project Files to an SCC Provider Project

The `Project -> Settings -> Members` dialog also has a feature for adding filenames from a Hawk project file to an SCC Provider Project. To do this, complete the following steps:

1. Make an SCC Project association using the `Setup` button (as described in the preceding topic [Associating Version Control Projects with Hawk Projects](#)).
2. Verify that a check box at the bottom of the tab becomes enabled, which will `Add Project Files to VCS Project`.
3. Click `OK` to close the `Project -> Settings` dialog; each Hawk project file will be added to the SCC Provider Project. A prompt will most likely come up asking for optional comments for the files to be newly checked in to version control.

Hawk projects make version control more visual in Hawk. When files are a part of a Hawk project they are displayed in Hawk's `Project Window` in the `File View` tab. The icons that represent files in the `Project Window` are displayed differently depending on what state the file is in. These are shown below.

-  When the files are checked-in (or read-only) the icons that represent the files will be gray with horizontal lines in them.
-  When the files are checked out Hawk conveniently places a red check mark to the upper right of the file icons.
-  If another user has a lock on a file that is part of the project being worked on, Hawk represents the file with a gray check mark in the `Project Window`.
-  Some Version Control projects are set to delete work files when they are checked in. Hawk represents files not found at the specified location (usually archived) with an exclamation point.
-  If the file currently being edited (as it is stored on disk) differs from the latest revision in version control, Hawk will display the file's representative icon in a pale yellow color, indicating that it is an old version.
-  When using the `Project Window` to carry out version control operations, or view version control status, it is a good idea to press the refresh button  from time to time to see the most current status of the files.

The file-icons in the `Project Window` help the user visualize the various states that files can be in when they are under version control.

Current Project Tree List

Hawk projects make the `Current Project Tree List` in the `Source Code Revision Control-Maintenance` dialog available. The `Current Project Tree List` displays the files in the current Hawk project and allows the dialog's

version control operations to be performed on one or more of those files. It eliminates the confusion of having to sort through unnecessary and unrelated files that may not be part of the project.

VCS and the User-Defined Popup Menu

Hawk features a variety of popup menus that can be customized. You may already be familiar with one of these, the menu that appears when you click the right mouse button. The right-mouse-click popup menu is context-sensitive. That is, it changes depending on where the pointer is when the mouse is being clicked. The popup menu discussed in the following paragraphs comes up when the right-mouse button is clicked while the pointer is in the current document with nothing highlighted.

Modifying the Standard Popup: A Simple Example

The right mouse popup menu can be changed using Hawk's popup menu editor. The popup menu editor modifies menu description files that contain functions that form Hawk popup menus. Hawk's main menu description file is called `CWRIGHT.MNU`. The `Edit this menu` option available on most standard popup menus will access the popup menu editor.



More information about the popup menu editor is available in [Chapter 12 Customizing the Interface](#).

The main popup menu already contains two commonly used VCS commands: `checkin` and `checkout`. The `checkout` command in the menu uses the `Check Out w/ Lock` command, as defined in the `Version Control Setup` dialog. Following is a description of adding an item in the pop-up menu to perform a non-locking checkout.

1. Right-click in an un-highlighted, non-HTML document to access the popup menu, then click `Edit this menu`.
2. Using the existing `Check out` line as a frame of reference, make a new menu item by first highlighting the `Check out` line and then clicking `Insert -> Menu Item`.
3. In the `Menu Item` dialog, type `Check out--no lock` in the `Menu Item Text` field. In the `Execute Function` field, type `MenuCommand IDM_TOOLS_VCS_GET`.

You can choose to have your new menu item inserted before the `Checkout` line by checking `Insert Before`.

4. Click `OK`, and then save the changes in the popup menu editor by clicking `File -> Save`.
5. Click `OK` to close the dialog.

The right-mouse button popup menu should now contain three VCS commands.



The above example works with command line or SCC integration for a version control system. The commands make calls via the provider `.DLL` or command line utilities, depending upon which system is currently in use.

Making Your Own Version Control Popup Menu

A custom popup menu can be made in Hawk with the popup menu editor. Use Hawk's default menu description file to store the new menu, or make a new one.

Example

To create a popup menu entitled PVCS, click **File** -> **New Menu** in the popup menu editor, then type the name **PVCS** in the **New Menu** dialog. This will add a [PVCS] section to Hawk's menu file, CWRIGHT.MNU.

The command to show a menu defined in CWRIGHT.MNU is DlgMenuPopup. Its syntax is described in Hawk's online Help. This command may be assigned to a keystroke, a button, another menu item, or a mouse click.

Example

If you use the **Key Bindings** dialog to assign <shift-mouse_right_click> to DlgMenuPopup=" [PvcS] ", you can call up your custom command menu by holding down the S key and clicking the right mouse button.

Using Multiple Configuration/Project Files (DOS VCS Utilities Only)

PVCS and many other version control systems use configuration/project files that maintain workfile and archive locations and other version control configuration information. When using DOS version control utilities, it is sometimes desirable to use more than one version control configuration/project file at a time. It may also be necessary to select the configuration/project file at the time a file is checked in or out. For example, one of your work files may be used in multiple projects.

The following examples give an idea of how multiple version control configuration/project files can be used.

Examples

If you need to access to two checkin commands without changing projects, you can set up each command with its own keystroke, button, or menu item.

1. Create a section in your CWRIGHT.INI file for each command:

```
[VCSCheckIn1]
CheckInSetCmd='put -cvcs1.cfg -M%Q %r%e'
CheckInBuffer
```

```
[VCSCheckIn2]
CheckInSetCmd='put -cvcs2.cfg -M%Q %r%e'
CheckInBuffer
```

The Hawk command ConfigFileRead can be used to execute the Hawk commands contained in these sections.

1. To bind the first checkin command to a button or menu item, use the function below:

```
ConfigFileRead='',VCSCheckIn1
```


2. For the second `checkin` command, use the function below:

```
ConfigFileRead=' ',VCSCheckIn2
```

If you are binding to a keystroke, use double quotation marks instead of the single quotation marks shown above.

The same method works for `checkout` commands. Below are sample `CWRIGHT.INI` file entries for `checkout` commands:

```
[VCSCheckOut1]
CheckOutSetCmd=0,'get -cvcs1.cfg %r%e'
CheckOutBuffer=0

[VCSCheckOutLock1]
CheckOutSetCmd=1,'get -cvcs1.cfg -l %r%e'
CheckOutBuffer=1

[VCSCheckOut2]
CheckOutSetCmd=0,'get -cvcs2.cfg %r%e'
CheckOutBuffer=0

[VCSCheckOutLock2]
CheckOutSetCmd=1,'get -cvcs2.cfg -l %r%e'
CheckOutBuffer=1
```

Version Control Integration Configuration

Hawk can work with version control software either through command line integration or the SCC Application Programming Interface (API). The command line integration works specifically with DOS command line utilities that perform version control operations (for example, PVCS `GET.EXE` and `PUT.EXE`, or MS `SS.EXE`). The SCC Application Programming Integration is for integrating GUI version control systems with the Hawk GUI.

The Hawk version control capabilities vary depending on which integration method is used and which Version Control Provider is used. The integration method is specified by selecting either the **Command Line Provider**, or the **SCC Service Provider** option in Hawk's `Tools -> Version Control -> Setup` dialog. Descriptions of the two integration methods follow. It is only necessary to use one integration method in order to access version control from within Hawk. Choose the method that applies to the type of version control system you have (SCC for GUI, or Command Line for DOS).

Version Control for Use with a Command Line Version Control Provider

Hawk is capable of supporting any version control system that can be run from a DOS command line. It comes with predefined commands for several version control systems, including Intersolv's PVCS, and various versions of the RCS utility, ported from UNIX.

Hawk integrates with command line version control utilities by creating a DOS shell to run the utility and then returning to Hawk once the task is complete. It is possible to perform the operation on a file currently open in Hawk. It is also possible to perform the operation on multiple files at once.

To use the Hawk command-line Version Control Interface, the Source Code Control System must be properly installed according to the instructions provided by the Version Control Vendor.

The following information is based on the assumption that command-line version control operations can be successfully performed from any directory in DOS according to the instructions provided by the Version Control Vendor.

To set up the version control command lines, complete the following steps:

1. Select the Hawk **Tools** menu.
2. Find the **Version Control** submenu.
3. Select the **Setup** item.
4. Make sure that the **Command Line Provider** radio button is selected.
5. Choose a command set from the list. You are now ready to use the commands on the **Tools -> Version Control** menu.

Adding a New Command Line Provider to Version Control Setup Dialog

If you do not see your provider listed in the **Command Line Provider** section of the **Version Control -> Setup** dialog, you will need to create your own command lines. If you know that one of the command sets listed is similar to yours, you may want to select that one, and use its commands as a starting point for your own. To start from scratch, select **Other** from the list and set up your own commands. If you don't see **Other** in the list, you can add it by adding **Other** to the line that looks like the following:

```
_StateHistory=XVCS,RCS,TLIB,PVCS,SourceSafe
```

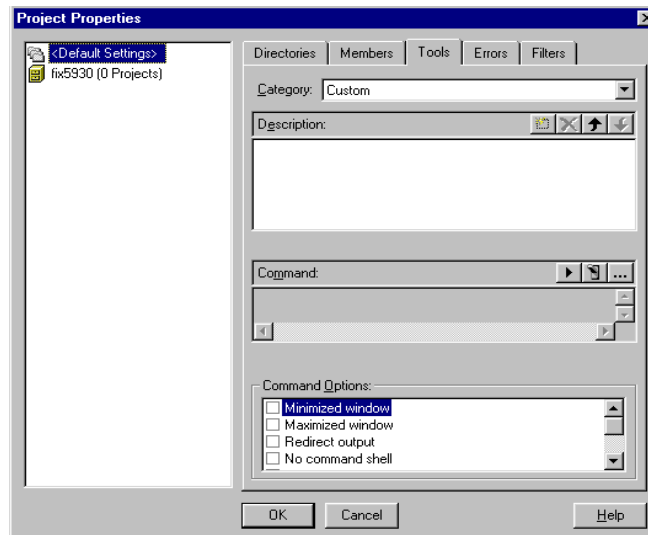
This line is located in the **[Editor]** section of Hawk's configuration file, **MWHAWK.INI**. The **MWHAWK.INI** file can be found in the Hawk home directory.

Customizing Version Control Commands

Whether you use existing version control command lines in Hawk or make your own, you will need to use the **Project -> Settings** dialog to define or modify them. To do this, complete the following steps:

1. In the **Tools -> Version Control -> Setup** dialog, choose the **Configure** button. The **Project -> Settings -> Tools** dialog, as displayed in [Figure 9-3](#) is activated.

Figure 9-3. Project -> Settings -> Tools Dialog



2. Select the entry **VCS** in the Category field.
3. Select the entry labeled **External VCS Check In** in the Description field.
4. View the command sequence displayed in the Command edit control. This is the command that is executed anytime you select **Check In** from any of the various menus and toolbars that provide the **Check In** item. The command includes Hawk filename component macros.
5. To see exactly how the command will run, select the **Test** button (switch icon). Note that when you have a dummy file such as `FOO.C` opened in Hawk, the macros resolve and the command is displayed for operation on the current document, `FOO.C`.

A description of some of the default commands listed in `Project -> Settings -> Tools` for version control are described below.

Default Command-Line Version Control Commands Described

The examples contained in this section use PVCS version control commands, but the concepts apply to any command-line version control provider.

Check-in Command

The `check-in` command is the command line needed to check a file into your version control system or archive. If possible, this command should work whether or not an archive currently exists for the workfile.

- If the `check-in` command is empty, the following command is used:

```
Put %b%e
```

- The initial `check-in` command defined in your `MWHAWK.INI` file is as follows:

```
Put -t@%Q -m@%Q %b%e
```

- The `@` sign precedes the name of a message file that contains the change description you want to use. This can be any text file up to 64K in size.

- `%Q` is a Hawk filename component macro that expands out the name of a temporary response file containing the text from the `Comment String (Check-in)` or `Additional Options (Check-out)` edit box. This macro is case sensitive. You must use an upper-case `Q`.
- `%b` is a Hawk filename component macro that expands out the complete workfile name, less the extension.
- `%e` is a Hawk filename component macro that expands out the extension of the workfile. This component begins with a dot (`.`) unless the extension is null.
- The additional flags and `%Q` macro enable you to supply a description of the changes or of the archive itself, when creating a new archive, to the `Check-in` command. This is done through prompting, or through the `Comment String` edit box in the `Check-in` dialog. The text is placed in a temporary response file.
- If you want to supply all parameters to the command, including filenames, from within the temporary response file the `%Q` macro creates, use the form:

```
Put @%Q
```

Predefined command lines are provided for several version control vendors. You select between these providers on the `Tools -> Version Control -> Setup` dialog.

Check-out Command

The `Check-out` command is the command line to execute when you are checking out a revision from a version control system or other archive. If your version control system offers revision locking, this command should not lock the revision. This is the command to be used when you are browsing or compiling the file, rather than planning to change it.

- If you don't define a command for `Check-out`, the following command is used:

```
Get %b%e
```

- If you want to be prompted for additional options or to supply them through the `Additional Options` edit box in the `Check-out` dialog, use a command similar to the one below:

```
Get @%Q %b%e
```

Predefined command lines are provided for several version control vendors. You select between these in the `Tools -> Version Control -> Setup` dialog.

- Use the `Check-out with locking` command to check out a revision for the purpose of changing it. If your version control system does not offer revision locking, you may either make this command the same as the `Check-out` command or leave it blank.
- If you don't define a command for `Check-out with locking`, the following command is used:

```
Get l %b%e
```

- If you want check-out with locking, and wish to be prompted for additional options (or to supply them through the `Additional Options` edit box in the `Check-out` dialog) use a command similar to the one below:

```
Get 1 @Q %b%e
```

- Predefined command lines are provided for several version control vendors. You select between these in the `Tools -> Version Control -> Setup` dialog.

Lock Command

The `lock` command is the command that your version control system uses to lock a revision without checking it out. This is useful when you already have a modified version of the source file that you want to check in, but you discover that the file was not locked.

Review each of the External VCS commands on the `Project -> Settings -> Tools` dialog. Note the commands as provided are designed to minimize response required from the user during command execution. Revise the commands as necessary to accommodate your particular source code control operation standards.

Additional Tips

Refer to the following tips regarding command line Version Control integration:

- The version control command runs in a DOS shell. Once the command is executed, the shell exits and any output that would normally be displayed in DOS is lost. In order to see the output, mark `Use VDOS` for each of the Version Control commands on the `Project -> Settings -> Tools` dialog. The output will then be displayed in the `Build` tab of Hawk's `Output Window`.



By default, the `Output Window` is docked on the bottom edge of the Hawk window. If it is not visible, go to the Hawk Window menu and check the `Output` option.

- One of the most commonly reported errors is `Bad Command or Filename`. Placing your version control provider in your path should eliminate this error.
- The command line provider names displayed in the `Version Control Setup` dialog are read from the following line in your Hawk configuration file, `MWHAWK.INI`. This line is typically placed in the `[Editor]` section in `MWHAWK.INI`:

```
StateHistory=XVCS,RCS,TLIB,PVCS,SourceSafe
```

`XVCS` is the name of the list internal to Hawk. This key is required. The names in the list follow the `xvcs` keyword on the line. Add to or edit this line as appropriate.

Hawk SCC Integration with Version Control Systems

Hawk provides a configurable interface for use with your Source Code Control System. Hawk does not come with a source code control system of its own. You must have a working source code control system to successfully use the interface.

All of Hawk's SCC Provider support is based on the Microsoft Common Source Code Control (SCC) Specification. This manual describes a dozen or so entry-points that

cover various version control operations (such as `check-in`, `check-out`). For SCC Providers to comply, they must provide an SCC Server DLL that contains these entry-points. When this SCC Server DLL is properly installed and configured, IDEs such as Hawk can connect their dialog boxes to these entry-points and get similar functionality, independent of SCC Provider.

Three examples of compatible SCC Server DLLs include the following:

- Intersolv calls their SCC Server DLL, `PVCS_API.DLL`, the "PVCS Version Manager Interface for Microsoft Development Environments." It comes on the same CD as `Version Manager`, and has a separate installation procedure.
- Microsoft Visual Source Safe has `SSSCC.DLL`, which should be ready to use from within Hawk once Microsoft Visual Source Safe has been properly installed.
- Hawk also provides its own SCC provider DLL for PVCS named `PVCSIF.DLL`. It is routinely called the "PVCS Developers Toolkit". `PVCSIF.DLL` provides low-level version control functionality which, in most cases, is more limited than Intersolv's DLL. `PVCSIF.DLL` calls into Intersolv's `VMWF520.DLL`.
 - Intersolv's `PVCS_API.DLL` didn't exist when Hawk's initial SCC interface was implemented (Hawk 1.0), thus explaining the reason for the existence of Hawk's `PVCSIF.DLL`. Hawk continues to support this DLL, as some customers prefer it over Intersolv's offering.

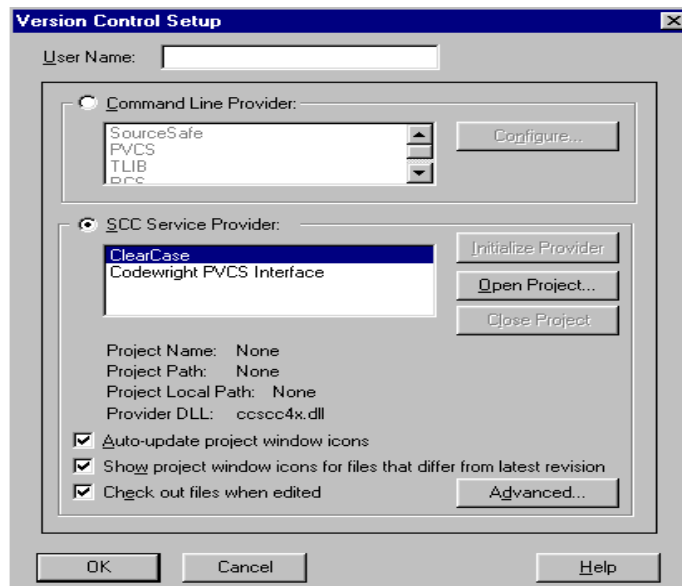
To find out whether a particular Version Control Provider offers a Microsoft Common Source Code Control (SCC) Specification DLL, check with the vendor.

Version Control for Use with a Source Code Provider DLL

Make sure you have a properly installed Source Code Control System on your computer. Next, complete the following steps to configure SCC Provider support:

1. Select the **SCC Service Provider** radio button on the `Tools -> Version Control -> Setup` dialog.

Figure 9-4. Version Control Setup Dialog



2. Select your version control system from the list in the dialog.
3. Press the **Initialize Provider** button.
4. Press the **Open Project** button in the Version Control Setup dialog.
5. Select the version control project you wish to use (the project should already exist in your version control system).

At this point you should be completely set and ready to perform check-ins, check-outs, and other version control operations from within Hawk.

If any of the above steps did not go as planned, read the following paragraphs.

Location of the SCC Provider DLLs

System information for SCC Version Control integration (the name and path of installed source code control systems accessible to your computer) is placed in the Registration Database for 32-bit systems.

If the Registry entries have not been made, or have been made incorrectly, it will not be possible to initialize the Version Control Provider in Hawk's Tools -> Version Control -> Setup dialog. The most common message that comes up when the initialization fails is System or Project Initialization Failed.

If this message occurs, there are a couple of courses of action:

- Make sure all version control integration modules provided by the version control vendor are properly installed. Many of the integration modules are not installed with regular installations of the version control system, and some of the integration modules don't even come on the same CD. The version control vendor should be able to provide the necessary information for where to get version control integration modules and how to install them.

- If the necessary version control integration components have been properly installed, but Hawk integration initialization still fails, it may be necessary to check the registration information for the integration modules in the Registration Database. Examples of some typical registry entries for version control integration can be found in Hawk's online help, under the topic *Version Control Configuration*. Compare these entries with similar entries in your Registration Database, and then verify that the paths to the various indicated DLLs or other modules are correct, and that the DLLs or modules do, indeed, exist.



The source code control vendor should make the appropriate system entries as part of the installation process. If the Source Code Control System is installed on a network server that is accessed by client machines, it is possible that the proper entries will not have been placed in the Registration Database of client machines. Sometimes VCS vendors provide network setup options in anticipation of such situations.

10

Search and Replace and Other Navigational Tools



This chapter describes the following Hawk code navigation tools:

[Search and Replace and Regular Expressions](#)

[Selective Display](#)

[Bookmarks](#)

[Button Links](#)

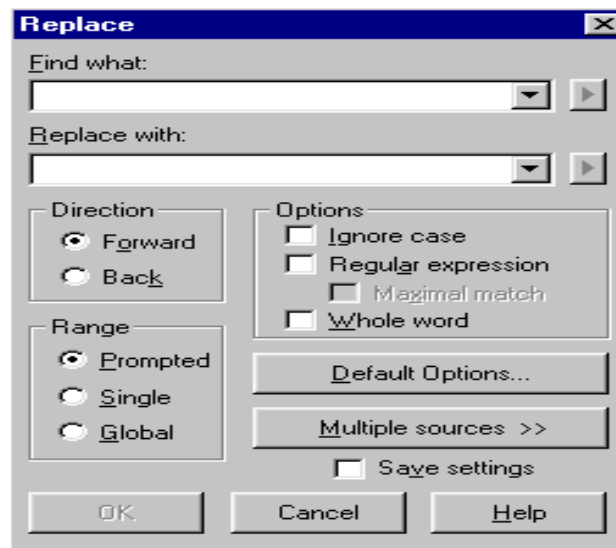
Search and Replace and Regular Expressions

Hawk offers several methods for performing search and replace operations. Each has its own advantages under differing circumstances. Some work on a single and others work directly on the files on disk. Some are meant to be quick and simple while others are designed for power.

Search and Replace Dialog

Many of the search features can be accessed from the `Search` menu. Clicking the `Replace` option on the `Search` menu accesses the `Replace` dialog. The `Search` dialog has the same options as the `Replace` dialog, excluding the replacement options. It is displayed in [Figure 10-1](#).

Figure 10-1. Replace Dialog



Search and Replacement Edit Boxes

The `Find what` and `Replace with` edit boxes allow you to enter the pattern you want to match and the replacement string. Both of these edit boxes maintain a history of previous responses, which you may select by pressing the down arrow to the right of the edit box.

Save Settings

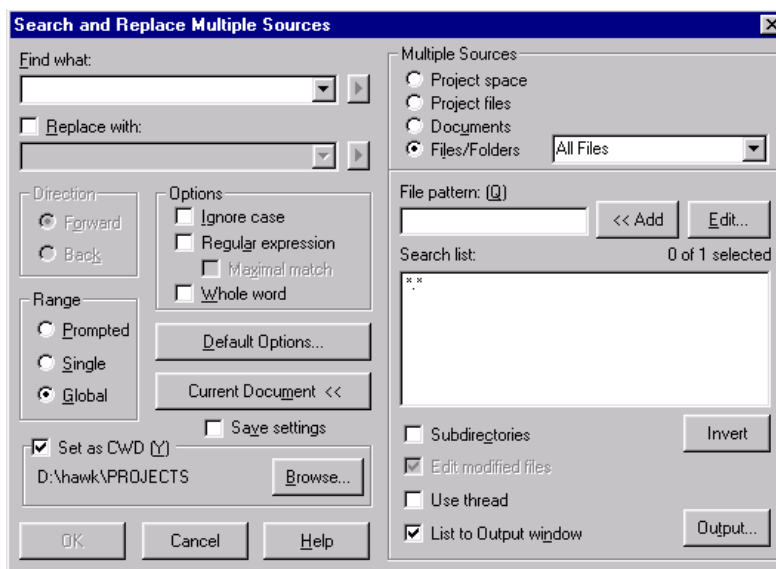
The `Save settings` check box is provided to save you from editing the `Search Options` dialog anytime you want to make a change to the search values. Settings are not immediately written to disk, however. Checking the box makes the current settings the default for that search session.

Multiple Sources Search Dialog

The `Search and Replace Multiple Sources` dialog allows you to define a search or replace operation that is not confined to a single document. You may elect to

search through all loaded documents, through files that are members of the current project space or project, or through any arbitrary set of files. To access the Search and Replace Multiple Sources dialog, click **Multiple Sources** on the Search menu.

Figure 10-2. Search and Replace Multiple Sources Dialog



Search and Replacement Edit Boxes

The Find What and Replace With string edit boxes in the Search and Replace Multiple Sources dialog allow you to enter the pattern you want to match and the replacement string, just as you would in a single document search. Replace is disabled until the Replace With option is checked. Both of these edit boxes maintain a history of previous responses, which you may select by pressing the down arrow to the right of the edit box.

If you elect only to search a series of files, rather than selecting the Replace action, you are selecting to perform a File Grep, which can also be done from the File Grep dialog (Search -> File Grep). A list of matches in the specified files will appear in the Search tab of the Output Window:

- The window also displays the line numbers and the line containing the matching text.
- To move to any of the matches listed in the message box, double click on it with the mouse.

Several features exclusive to the Search and Replace Multiple Sources dialog make it one of the more convenient navigating tools available in Hawk. Descriptions of some of these features are provided below.

Current Directory

The directory defined for the search is displayed at the lower left of the Search and Replace Multiple Sources dialog, under the check box labeled Set as CWD. Items to consider with regards to the current directory include the following:

- **The Browse Button:** If you are not in the desired directory, press the **Browse** button to select a new working directory for Hawk.
- **Set As CWD:** After changing the directory, you may want that directory to remain in effect after you finish searching and return to editing. If this is so, check the **Set As CWD** box. If you just want the directory in effect for the current search, ensure that this box is not checked.

Multiple Source Options

Hawk has the following options in the **Multiple Sources** Group in the **Search and Replace Multiple Sources** dialog:

- **Project Space:** Use this option to search the members of the current project space. A list box displaying all projects in the project space will appear when the option is selected. All projects are initially selected for searching. Choose only those that you wish to search.
- **Project:** Use this option to search through members of the current project.
- **Documents:** Use this option to search the list of currently loaded documents only.
- **Files and folders:** Use this option to search an arbitrary group of files in any series of folders. Enter a name for your search set in the edit box associated with this radio button, or select from the list of previously created search sets.

File Pattern

You may use the **File Pattern** edit box in the **Search and Replace Multiple Sources** dialog for quick, ad hoc searches that you are not apt to repeat. It lets you specify a file type or series of file types to search without associating it with a search set name. While not as powerful as the **Search list**, it is simpler to use. Enter one or more wildcard patterns into this box. Separate multiple patterns with a semicolon.

The **File Pattern** supplements the selected items on the **Search list**, if any. You may add it to the **Search list** by pressing the **Add** button.

- The **File Pattern**, when specified, does not override the list of files in the **Search list**. If both are defined, the files in both the **File Pattern** and any selected members of the **Search list** are searched.

Search List

The initial contents of the **Search List** box in the **Search and Replace Multiple Sources** dialog depend on which mode of search is selected.

- If **Project space** is selected, the box lists all files in all selected projects.
- If **Project Files** is selected, the box lists the files in the current project.
- If **Documents** is selected, the box lists the documents currently open in Hawk.
- If **Files/Folders** is selected, the box reflects the search set shown in the **Files/Folders** combo box.

- The `Search List` is usually a series of wildcard patterns. To edit the list, press the **Edit** button to bring up the `Edit Files/Folders` dialog, described under the topic [Edit Search List](#).

When you first select your desired mode, you will see that the entire contents of the list box are selected. If you wish to further limit the search or replace to a subset of the list, deselect the files you want to exclude.

Selecting Files from the Search List

Click a filename or wildcard pattern in the `Search and Replace Multiple Sources` dialog's `Search List` to select it for the search operation. Use the following selection options:

- Select or deselect additional files or patterns by holding down **Ctrl** clicking on those members of the list.
- Select all files and click the **Invert** button to start your selection process over again.
- If the list of documents you do not want to search is shorter than the list of documents you *do* want to search, select the documents you want to leave out and then press the **Invert** button. All of the documents that were previously selected become unselected, while the unselected documents become selected.

Search Subdirectories

Check the **Subdirectories** box in the `Search and Replace Multiple Sources` dialog if you want the search to encompass files in subdirectories of the path searched.

Edit Modified Files

If you are performing a replace operation, you may wish to review the changes made, or at least to know which files were changed. Select the `Edit Modified Files` option in the `Search and Replace Multiple Sources` dialog to do this. Modified files are loaded in Hawk for viewing and possible editing. This is very convenient if you plan to check the modified files into version control.

Threaded

The `Search and Replace Multiple Sources` dialog's `Use Thread` option allows the search or replace operation to proceed as a separate process, so that other tasks can be performed as the search continues. If the operation is a search only (`File Grep`), the `Output Window` updates as matches are found. Otherwise, matches are listed after the operation completes.

Send Listing to Output Window

The `Search and Replace Multiple Sources` dialog's `List to Output Window` option controls whether and what information is sent to Hawk's tabbed output window. Its primary purpose is to let you see the results of Search only (`File Grep`) operations, but it can also be useful as a summary of replacement operations.

Press the **Output** button to display the following options in the `Search Output Options` dialog:

- `List filenames only`: Normally, search output contains the file and line numbers of matches, and the line of text in which the match was found. For a list only of filenames in which matches were found, check this box.
- `Append to listing`: To add to the information in the output window rather than replace what was there from previous searches, check this box.
- `List to file`: You may specify that the information about the matches found be stored in a file; you may also specify the name of that file. The default name is `CWFGREP. ____`, which is created in the current directory. You may specify another name if the default causes a conflict, or if you wish to save the output from the previous or current operation, rather than allowing it to be overwritten. You may also redisplay the results of a previous search by specifying the name of the file in which the results were saved.

Edit Search List

The `Edit` button in the `Search and Replace Multiple Sources` dialog brings up the `Edit File/Folders` dialog used for editing the `Search List` more specifically. The dialog's options are as follows:

Search Pattern

For the `Edit File/Folders` dialog's `Search pattern` edit box complete the following steps:

1. Enter the file specification you wish to add to the current `Search Set`.
2. Press **Add** to add the file pattern to the `Patterns` list. Standard wildcard characters `?` and `*` are allowed. Your pattern may be anything from `*.c` to `k:\src\cw????.*?v`. You can enter several patterns at once by separating them with semicolons. When you do so, each pattern is given a separate line in the list of patterns for the `Search Set`.

Patterns List

The `Edit Files/Folders` dialog's `File Pattern` list box contains the patterns already defined for this `Search Set`. The `Search Set` is composed of the union of these patterns, rather than the intersection. Files are only searched once, even if they match several of the `Search Set`'s patterns. Therefore, if the `Search Set` contains the pattern `*.*`, other patterns are superfluous unless they contain a path element. Source patterns that do not contain a path element will apply to the current directory.

Drive and Directory Lists

The drive list box and the `Directories` tree list in the `Edit Files/Folders` dialog allow you to select the path you want to apply to the source set. This is only meaningful if the `Include Directory` check box is checked when the `Search Pattern` is added to the `Patterns` list.

Include Directory

When `Include Directory` is checked in the `Edit Files/Folders` dialog, the drive and directory selected in the adjacent list boxes are automatically added to the pattern specified in the `Search Pattern` edit box as you press the `Add` button.

List Editing Buttons

The List Edit buttons in the `Edit Files/Folders` dialog work as follows:

1. Use `Add`, or `Delete` to add or remove members of the `Search Patterns` list.
2. Use `Invert` to reverse the current selection; those items that were selected become deselected and visa versa.
3. Use `Clear` to deselect all the patterns on the list so that you can start selecting from scratch.

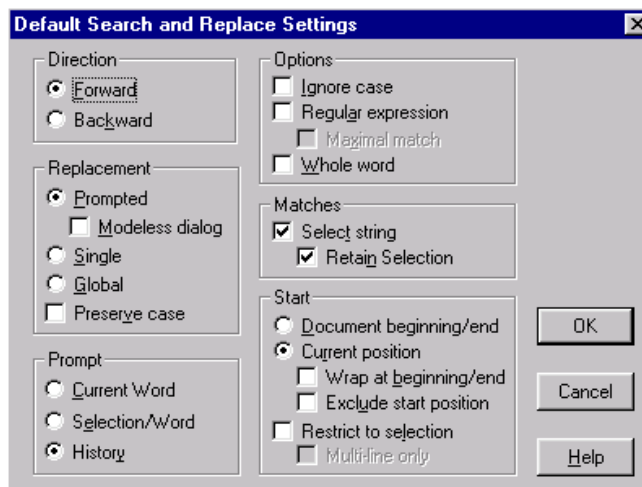
Default Button

All of the search dialogs (with the exception of `File Grep`) have a `Default Options` button. The `Default Options` button accesses the `Default Search and Replace Settings` dialog, described next.

Default Options

Use the following dialog to set default options for search and replace activities.

Figure 10-3. Default Search and Replace Settings Dialog



To access the `Default Search and Replace Settings` dialog, complete one of the following:

1. Click `Options` on the `Search` menu.
2. Click `Default Options` in the `Search` dialog.
3. Click `Default Options` in the `Replace` dialog.

4. Click **Default Options** in the Search and Replace Multiple Sources dialog.

The items in the Default Search and Replace Settings dialog are listed next.

Search Direction

The search Direction section of the Default Search and Replace Settings dialog allows you to search forward from the cursor position or backward.

Range

The Range group of options in the Default Search and Replace Settings dialog is specific to the replacement operation.

- **Prompted replacement.** When the Prompted radio button is selected, you are prompted each time text matching the search pattern is found. You may elect at that time to make or skip the replacement, or to cancel the search. The search continues until no more matches are found in the defined scope of the search, or until you select cancel. Enabling this option consequently enables the **Modeless dialog** option.
- **Modeless dialog.** This option works with single documents only. It works in conjunction with a prompted replacement. When Hawk prompts with options to **Replace Current**, **Find Next**, **Replace Global**, and so forth, the **Modeless** option eliminates the necessity of canceling the prompt in order to edit the document. With the **Modeless** option turned on, the prompt can be left running at any point that direct editing of the current file becomes necessary. When one is ready to proceed with the search/replace, the prompt can be re-accessed without restarting the search.
- **Single replacement:** When you select the **Single** radio button, the first occurrence of matching text is replaced without prompting.
- **Global replacement:** Select the **Global** radio button to cause all occurrences of matching text within the scope of the search to be replaced without further prompting.
- **Preserve Case**—applies to Search/Replace: The **Preserve Case** option sets the case of the replacement string to match the case of the search string. The text must follow any of the following patterns:
 - all upper case
 - all lower case
 - capitalized (first character upper case with the rest in lower case)
- If the matching text is found to be in mixed case, the matching text will be replaced with the same case as the string that was entered in the **Search/Replace** dialog.

Prompt

The options in the `Prompt` section of the `Default Search and Replace Settings` dialog control what word appears in the `Find What` field by default. Definitions of the options are as follows:

- `Current word` uses the word at the cursor as the default search string.
- `Selection/Word` uses the contents of the selection as the default search string. If there is no selection, use the word at the cursor.
- `History` uses the most recent response in the history list (combo-box drop-down).

Search Options

The `Options` group of check boxes is available in both the `Search` and the `Replace` dialogs, as well as the `Default Search and Replace Settings` dialog. It is also available in the `Search and Replace Multiple Sources` and `File Grep` dialogs. The options allow you to turn various search attributes on or off. These include:

- `Ignore case`: When this box is checked, uppercase characters or lower case characters will be matched. When it is not checked, the case of the characters in the specified search string is significant.
- `Regular expression`: When checked, this box indicates that the search pattern is a regular expression. Some of the characters in regular expressions are given a different meaning than they would have in an ordinary string search. This allows for more powerful searches.
- `Maximal match`: A check in this check box indicates that regular expressions should match the largest possible unit. If this box is not checked, regular expressions will match the smallest possible unit, which in some cases may be 0 characters. For example, if the regular expression specifies matching 2 or more A's in a row (`AA+`), and the search encounters 5 in a row (`AAAAA`), what does it match? If maximal match is on, it matches five. If it is not, the search matches the first two.
- `Whole word`: When the `whole word` box is checked, the pattern you are searching will not match strings that are only partial words. That is, the pattern must be preceded and followed by one of the following:
 - beginning of file or end of file
 - beginning of line or end of line
 - spaces or tabs

Matches

The following settings are under the heading `Matches` in the `Default Search and Replace Settings` dialog:

- `Select string`: The `Select String` check box specifies whether the text that matches the search will be highlighted in a selection at the end of the search.

The selection may be momentary, or may be retained so that you can operate on it (copy, cut, or replace).

- **Retain selection:** When the `Select String` box is checked, the `Retain Selection` box indicates whether the selection that encompasses matching text is momentary or retained so that you may operate on it.

Start

The following options are listed under the heading `Start` in the `Default Search and Replace Settings` dialog:

- **Document Beginning/end:** When enabled, the `Document Beginning/end` option causes the search to go through the whole document regardless of cursor position.
- **Current Position:** When enabled, the `Current Position` option causes the search to go from the cursor position to the end of the document. Choosing this option enables the `Wrap at beginning/end` and `Exclude Start Position` options:
 - **Wrap at beginning/end:** A check in the `Wrap at beginning/end` check box indicates that you want to search the entire document. When the document extremity is reached (the beginning or end of the document, depending upon the direction of search), the search is continued from the other document extreme. The search concludes at the point at which the search began.
 - **Exclude Start Position:** The `Exclude Start Position` option tells Hawk that if there is text matching your search at the current cursor position to ignore it. Presumably, you are not searching for this text, and perhaps have just finished editing it.
- **Restrict to selection:** The `Restrict to Selection` check box allows you to indicate whether you want the search to be restricted to the selection or marked block of text. If the box is not checked, the scope of the search is global. This option will be disabled if no selection is defined. Enabling this option consequently enables the `Multi-Line Only` option.
 - **Multi-Line Only:** This option is a modifier for the `Restrict to Selection` option. When enabled, the search will only be restricted to the selection if the selection spans more than one line. This eliminates incidental selections of a single word, and most selections resulting from a search.

Example: Multi-Source Search

An example of one of the ways the Hawk `Multiple Source Search` dialog can be used to save valuable time follows:

To look for the string version in two directories, `C:\DOCS*.TXT` and `D:\PROJECT*.C`, complete the following steps:

1. Select **Search -> Multiple Sources** to get to the multiple-source search dialog.

2. Select the **Files/Folders** radio button in the upper-right corner of the dialog.
 3. In the accompanying combo box, select **All Files**.
 4. Click the **Edit** button to bring up the **Search List Edit** dialog. If you just want to do a search through files in the current directory, you can skip this step, and just type in the file spec, with wildcards, into the **File pattern** box.
 5. Use the **Directories**, **Drives**, and **Files** lists to select the **C:** drive and the **\docs** directory.
 6. A list of files in that directory appears in the **Files** list. You can restrict the number of items in this list by selecting the appropriate entry in the **Filters** list.
 7. Highlight the files you want to search, then hit the **Add** button in the lower portion of the dialog.
 8. Repeat steps 5-7 for the **D:\PROJECT** directory. The **search list** should now show the files you want to search.
 9. Click the **Invert** button to highlight the list, or select just the specific files you wish to search.
 10. Click **OK**. The output window will appear, and you will see results of the search as the search pattern is found in the target files.
- The **List to Output Window** box has no effect on search-only operations. You would check this box if you were doing a search/replace and wanted to see a report of what replacements were made.

This may seem like a lot of mouse clicks. But often you will find that the last settings you used are the ones you want, so most of these steps can often be skipped.

Incremental Searching

Incremental searches can save time and typing. This is because the string is matched as the word is typed, instead of being matched after the word is typed. Many times the search function finds the string you are looking for before it is all typed in. Hawk's incremental search function is called `ISearch`.

ISearch Function

The Hawk `ISearch` function is for incremental searching. It may already be bound to a key in your keymap, but if not you can easily add such a binding with the **Keyboard** dialog on the **Customize** menu.

You begin the search by invoking `ISearch`; press **CBi** in the CUA keymap, for example. `ISearch` prompts you for the search string, and you begin typing. Keep in mind that `ISearch` always searches in a case-sensitive fashion. The case of the characters you type must agree with the case of the characters in the document, in order to match.

When you type the first character, `ISearch` moves to the next occurrence of that character and highlights it. `ISearch` performs a search after each character you type, looking for the sequence of characters you have typed thus far. If it doesn't find a


match, it issues a beep, and the character you just typed is removed from the search string.

If you make an error while you type, just backspace over the incorrect character or characters. As you remove characters from the search string, the cursor backs up to the position that first matched the characters that remain. Delete the entire search string and you will find yourself at the position where the search began.

You cancel `ISearch` by pressing `X`. At that time, the string you typed into `ISearch` is added to the search response history to make searching for that string again more convenient. You may also use the `Quick Search` feature to search for the word that is now at the cursor position.

Quick Search

The `Quick Search` facility is a function you can use to do an immediate search for the word at the cursor. In the default configuration, you will find this function assigned

to the  button on the `Standard Toolbar` and assigned to the `CBq` key combination in the CUA keymap. There is also a `Quick Search` option in the right-click pop-up menu that starts a quick search. Other assignments may be made, or readily changed.

Toolbar Search

"Toolbar Search" refers to the search capability built into the `Standard Toolbar` in the form of a drop-down list box control. The toolbar search provides the most immediate, convenient way to perform simple searches.

To use the `Toolbar Search`, click in the `Toolbar Search` edit box (in the `Standard Toolbar`) and type in the string you wish to search for. This may be a regular expression pattern, if you have regular expressions turned on in the `Default Search and Replace Settings` dialog. The `Toolbar Search` box honors all of the settings in the `Default Search and Replace Settings` dialog. As soon as you press `Enter`, the search commences.

If Hawk finds a match, the `Toolbar Search` will retain the focus. This gives you the opportunity to search again just by pressing `Enter` again. Pressing anything else will cause the `Toolbar Search` to lose the focus, and you are then able to edit at the position where the match was found.

The `Toolbar Search` maintains its own response history, to allow you to recall strings or patterns you previously typed. These are available when you select the down arrow to the right of the edit box which causes the history list to drop down.

Regular Expressions

Regular Expressions are powerful notations for matching string patterns. Hawk makes UNIX-Style Regular Expressions available for its search feature. Hawk also uses Regular

Expressions for interactively customizing Hawk's Symbols, makefile reader, and Visual Studio Workspace reader.



For more information on Symbols, review the topic on *Outline Symbols* later in this chapter. For more information on the makefile and Workspace readers, read [Chapter 7 Projects, Project Spaces, and Workspaces](#).

Investing a relatively small amount of time to gain some knowledge of Regular Expressions will pay back in the long run with the amount time saved when searching for string patterns.

A Regular Expression can be as simple as a single, literal character, like "a". You could search for the character "a" from Hawk's *Search* dialog, with *Regular Expression* marked, and you would be searching using *Regular Expressions*. Such simple expressions are usually a sub-expression of more complex regular expressions. In reading the descriptions that follow, an "expression" may mean a single character, a group of characters, or a class of characters.

Special Characters

Regular Expressions give special meaning to certain characters. Some are operators and some show grouping. There is also a method of representing non-printing characters, such as a tab, within Regular Expressions. All other characters just represent themselves, as they would in an ordinary string search.

The characters to which Regular Expressions give special meaning are called metacharacters. These characters and their meaning are shown in [Table 10-1](#).

Table 10-1. Special Characters

Character	Meaning
.	Matches any single character, except a new line
*	Matches any number of occurrences (even zero) of the expression that precedes it.
+	Matches one or more occurrences of the preceding expression.
?	Matches zero or one occurrence of the preceding expression.
[and]	Defines the beginning and end of a character class.
(and)	Defines the beginning and end of a group of expressions. Groups expressions into larger units and dictates precedence. Each group is also a Reference Group which may be pasted into a replacement string.
	Alternation. Allows matching the expression on the left or on the right of the operator.
\$	Matches the end of a line.
^	Two meanings: Matches the beginning of a line. Complement operator when the first character in a character class.

Table 10-1. Special Characters

\	Used for escaping metacharacters and non printing characters.
\c	The position in the pattern at which the cursor is placed at the end of a successful search.

Escape Sequences

There are times that you want to use a metacharacter as itself – without the special meaning that regular expressions give it. For example, you may wish to match a dollar sign, rather than look for the end of a line. To match a dollar sign you must escape or quote it. This is done by preceding the character with a backslash. This is true of all the metacharacters. To match a dollar sign, for example, you use `\$` in your Regular Expression, rather than just `$`.

Example

`cat$` matches the word `cat` only if it is immediately followed by a newline character.

`cat\$` matches the word `cat` only if it is immediately followed by the character `$`.

Table 10-2 is a list of other escape sequences supported by Hawk's Regular Expressions:

Table 10-2. Hawk Regular Expressions

Escape	Meaning
\n	New line (<CR><LF> or <LF>, depending on how it is defined for the document)
\t	Tab
\b	C - H (backspace)
\r	Carriage return
\f	Form feed
\nnn	Octal value between 0 and 0377.
\xnn	Hexadecimal digit between 0x00 and 0xFF
\m	The literal character m.

Matching a Character

The basic unit of a regular expression is matching a single character. You can match a single character in one of three ways:

- Literally, by using the character itself or the appropriate escape sequence. (such as `'cat'`)
- Ambiguously, by using the dot (`.`) metacharacter, if matching a character literally is too limiting. (such as `'..t'`)

- With a Character Class. If a literal character is too narrow a match and the dot is too broad a match, a character class can be used for anything in between (such as `[A-Za-z]`).

Character Classes

A character class is a series of characters enclosed in square brackets. It specifies a set of characters, any one of which may match. For example, the following character class matches any vowel, whether upper or lower case:

```
[AEIOUYaeiouy]
```

Ranges of characters may also be specified within a character class. This is done by placing a dash between the character that begins the range and the character that ends the range. The following character class matches any character between 0 and 9:

```
[0-9]
```

How do you match a DASH (-), then ? If it is not in the character range you specified, just place it at the beginning or end of the character class where it is not between two characters of the class. If you precede the dash with a backslash (\) you can put it anywhere within the class.

The CARET (^) has a special meaning when it appears as the first character of a character class. It complements the class. When it appears at any other position within the character class, it just adds the up-caret to that class. You may use it as a shorthand method of saying "match any characters except for the following:" rather than specifying a large character class.

Example

```
[^$. | () {} *+?^]
```

Matches anything except the eleven characters following the up-caret.

Escaping Characters in a Class

Metacharacters may be used in character classes without escaping. The only characters that must be escaped are as follows:

] " \ and sometimes - and ^ depending on the position within the class.

- Escape the - when you use it in the middle of a class but do not intend it to signify a range.
- Escape the ^ when it would otherwise be at the beginning of a class but you do not intend for it to signify complement.

When in doubt, escape the character. It cannot do any harm. The above characters look like this when escaped:

```
\] \" \\ \- \^
```

Iteration Qualifiers

Iteration qualifiers are metacharacters that are not regular expressions by themselves. Instead, they state how many iterations of the preceding expression there must be or can be, in order to match. These metacharacters are: `*`, `+` and `?`.

- The `*` matches any number of occurrences
- The `+` matches one or more
- The `?` matches zero or one.

Without these qualifiers, a regular expression will match exactly one occurrence in the text.

Examples

Let us consider some specific examples of how these qualifiers might be used in the task of matching white space:

- `\t*`
The example above will match any number of consecutive tabs including none. By itself, it is not very useful to match none of something. As part of a larger regular expression, it could be quite useful. For our purposes here, the following might be preferable:
- `\t+`
This example matches one or more consecutive tabs. The tab is represented as `\t`, and the plus sign says "one or more of the previous." To match white space, we need to match spaces too. We don't know what order the spaces and tabs will come in, and we don't know how many there will be. These are signs that we need a character class.
- `[\t]+`
The example above uses a character class containing a space and a tab. The `+` sign following it means that this Regular Expression will match any combination of spaces and tabs so long as there is at least one space or tab.

If you wanted to match the white space within a function call, where you knew there might be one space or tab, or there might be none, your expression could look like this:
- `\([\t]?`
The previous example searches for a left parenthesis followed by zero or one spaces or tabs. Since the left parenthesis is a metacharacter it is necessary to escape or quote it with a preceding backslash. The `\t` we have used before to represent a tab character. The question mark says "zero or one of the preceding."

Regular Expressions: Positioning at Beginning/End of Line

You may use the metacharacters `^` and `$` to qualify your Regular Expression further. We have already seen how the `^` may be used to complement a character class, but it has another quite different use outside of a character class. These metacharacters specify that, in order to match your Regular Expression, the text must appear at the beginning

or end of a line, respectively. Unlike the iteration qualifiers, however, these metacharacters can stand alone as Regular Expressions. That is, you can search for just the end of the line or just the beginning of the line.

Examples

Following are some examples of regular expressions that match strings at the beginnings and ends of lines.

- `^PUBLIC` matches the word "PUBLIC" when it occurs at the beginning of a line.
- `\) $` matches a right parenthesis, a character which must be escaped when it is found at the end of a line.
- `^\) $` matches a right parenthesis when it is the only thing on the line.

Alternation and Grouping

Alternation and grouping go hand in hand. Alternation often relies on grouping and is the primary need for grouping.

Alternation uses the metacharacter `|` to denote that a match has been found if the text matches either of two Regular Expressions. This operator may be repeated to indicate that the text may match any one of several expressions.

Because of the typical order of precedence, the alternation applies to the entire expression, if not limited by grouping. Grouping occurs when you place parentheses around one or more expressions that are part of a larger expression.

Examples

- `PUBLIC | PRIVATE` matches either "PUBLIC" or "PRIVATE".
- `PUBLIC (void | DWORD)` matches the word "PUBLIC", when followed by a single space, and either "void" or "DWORD".
- `^PUBLIC [\t]+ (void | int | long | DWORD)` matches, at the beginning of a line, the word "PUBLIC", followed by one or more spaces or tabs followed by any one of the following words: "void", "int", "long" or "DWORD".
- These examples begin to show the power of regular expressions.

Reference Groups and Replacement Strings

In addition to showing association and precedence, grouping allows the use of Reference Groups. A reference group is one or more expressions that have been placed between parentheses and then pasted into a replacement string by referencing its number.

Hawk assigns a reference number, from 1 to 9, to the text matching each group defined. Reference numbers are assigned from left to right. You may paste the associated text into a replacement string by using this number, preceded by a backslash.

The following demonstrates how reference groups may be used:

Example

Search string in document:

```
GotoXY(cat, dog)
```

Search pattern:

```
GotoXY\((.*) , (.*)\)
```

Replacement pattern:

```
move_cursor(\2, \1)
```

Replacement result in document:

```
Move_cursor(dog, cat)
```

Note that in replacement patterns you do not need to escape metacharacters, such as the left parenthesis, with a backslash. There are a few escape sequences that are meaningful in a replacement string, such as those used by reference groups, but such sequences are not themselves regular expressions. An operation using reference groups, such as that depicted above, could be used to reverse the parameter order used by one function when converting to a similar function.

There is an implicit reference group that you can use in replacement strings to represent the matching text in its entirety. You reference this group in the replacement string with an `&` at the desired location. This means that you must use `\&` to specify an ampersand in a replacement string, when regular expressions are enabled.

Placing the Cursor

When you search for a piece of text, it is usually because you are going to do something with it or to it. After a successful search operation, Hawk positions the cursor at the beginning of the matching text -- at least, that is the default action. What if you want to edit the other end of the matching text, or perhaps the middle?

When using Hawk's regular expressions, you have a special escape sequence available that allows you to indicate where in the matching text the cursor should be positioned. You should note that this is available only for search operations. It is not for use in replacement text.

The escape sequence that specifies cursor position is `\c`. An example of its use follows:

```
singletons\[\c.*\]
```

This example places the cursor at the beginning of whatever text is contained between the left and right square brackets. The square brackets are metacharacters and are therefore escaped. This cursor positioning facilitates editing the contents of the square brackets.

Examples

The examples in [Table 10-3](#) are intended to inspire your own uses of regular expressions:

Table 10-3. Regular Expression Example

Pattern	Description
<code>[A-Za-z_] [A-Za-z0-9_]*</code>	C language identifier
<code>-? ([0-9]+ \. ? [0-9]* \. [0-9]+)</code>	Floating point (real) number
<code>([\t] ^) [^ \t] +</code>	Beginning of word (simple white space)
<code>[^ \t] + \c ([\t] \$)</code>	End of word (simple white space)
<code>([^ A-Za-z0-9] ^) [A-Za-z0-9] +</code>	Beginning of word (Non-alphanumeric)
<code>[A-Za-z0-9] + \c ([^ A-Za-z0-9] \$)</code>	End of word (Non-alphanumeric)
<code>/* . * */</code>	Single-line comment (C language)
<code>/* . * \n ([\t] * * . * \n) * . * */</code>	Multi-line comment (C language)

Searching for Spaces, Tabs and other Blank Characters

One of Hawk's longstanding `Search/Replace` features is its ability to find and replace invisible characters. Parts of the topics below have been generally covered in the previous discussion on regular expressions, but they will be discussed more specifically with regards to spaces, tabs, and new lines, here.

Searching for Spaces or Tabs

To find tabs or spaces, complete the following steps:

1. Go to the `Search` pull-down and click **Search**.
2. In the `Find what` field type a space (press the space bar) to insert a space, or `/t` to insert a tab.
3. Before clicking `OK`, be sure to mark the **Regular Expression** and **Save Settings** options.

The **Regular Expression** option forces the search feature to search for non-literal characters and the **Save Settings** option applies the current search options to all search sessions.

Searching for New lines

To find new-line characters, complete the following steps:

1. Go to the `Search` pull-down and click **Search**.
2. Type `\n` in the `Find what` field.

- Before clicking **OK**, be sure to mark the **Regular Expression** and **Save Settings** options.

Regular Expression forces the search feature to search for non-literal characters, and the **Save Settings** option applies the current search options to all search sessions.

The procedure for replacing characters with tabs, new lines, and spaces is the same as the procedure for finding them, only the space, `\t`, and `\n` characters are typed into the **Replace With** field of the **Replace** dialog, instead of the **Find What** field of the **Search** dialog. The **Replace** dialog is accessed by clicking the **Replace** item in the **Search** pull-down menu.

Searching for Control Characters (binary/hex data)

You can use control characters in both search and replacement patterns. You can specify any byte (hex) value as part of a pattern.

First, turn on **Regular Expressions** from the appropriate dialog. Then use the `\x` notation to specify hexadecimal values. For example, to search for the form feed character, you can use the following string:

`\x0c` (backslash x zero c)

Make sure there are two digits following the `x`. If you want to search for two consecutive bytes by hexadecimal value, string them together like the following:

`\x0c\x0d`

Searching for New lines: Issues

Generally, using `\n` for matching or replacing line ends is the suggested method in Hawk (the `\` must be doubled when used in code), as described in the previous topic. In some cases, however, it is more desirable to search for new lines using their respective hexadecimal notations (for example, `0D0A`, `0A`, `0D`). Since hexadecimal `0D` and `0A` characters make up ends of lines in one form or another an exception has to be made when searching for and replacing them to avoid disrupting partial line-end sequences. Therefore, when searching for `0D` and `0A` hex characters, keep the following in mind:

- `\x0a` will match a solo (without a preceding `0x0D`) `0x0A` but not the pair.
- `\x0d` will match a solo (without a following `0x0A`) `0x0D` but not the pair.
- `\n` will match a solo `0x0A` or a `0x0D`, `0x0A` pair.
- Replacing `"\x0d"` with `"\n"` would be good for converting a Macintosh file (opened with auto-detect file type off).

Replacing `"\n"` with `"\n"` is a good way to convert a UNIX file to a MS-DOS file, or vice-versa, if the buffer flag for UNIX EOLs is set appropriately in **Customize** -> **Language** -> **Options**, or **Document** -> **Manager** -> **EOL/EOF**.

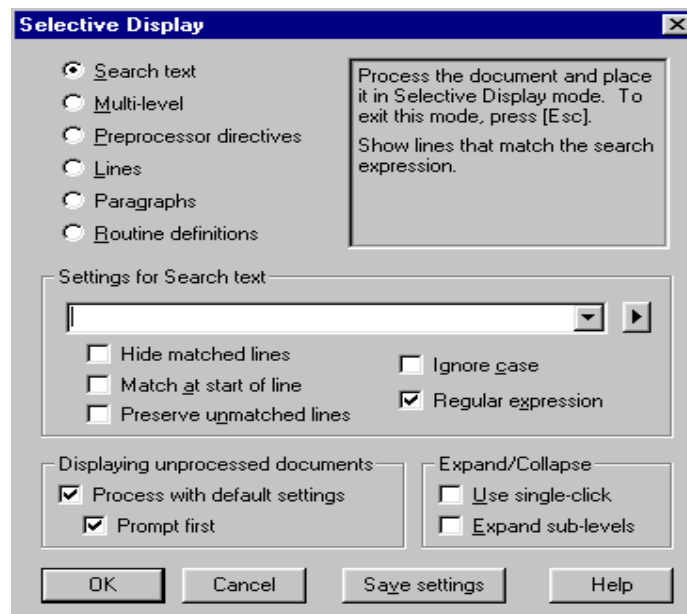


More information on setting and using UNIX EOLs can be found in [Chapter 16 UNIX](#).

Selective Display

Hawk's `Text` menu has an option for `Selective Display` (shown in [Figure 10-4](#)). `Selective Display` is a feature that "collapses" or "folds" text in the current document so that only desired text is displayed. The `Selective Display` menu item accesses a dialog that has options for controlling which lines of the document are visible.

Figure 10-4. Selective Display Dialog



The options in the dialog offer choices for how the text in the current document should be "collapsed". A quick description of the items in the dialog follows.

Selective Display Options

`Selective Display` options include the following:

- `Search text`: Displays or hides text matching a pattern.
- `Multi-level`: Hides text based on braces or indentation levels.
- `Preprocessor directives`: Hides text based on C/C++ language preprocessor directives.
- `Lines`: Displays or hides a selected group of lines.
- `Paragraphs`: Displays only the first line of each paragraph. A paragraph is defined as text separated from other text by one or more blank lines.
- `Routine Definitions`: Displays lines containing function definitions for the language indicated by the file type. In CUA, `CSg` also controls Routines selective display.
- Display on the right-mouse popup menu accesses many `Selective Display` options.

Pre-processed View

The `Preprocessor directives` option in the `Selective Display` dialog is a special option for C/C++ users who find their source files cluttered with `#ifdefs`. The option allows source code to be displayed exactly as the compiler will see it, while it is being edited.

Applications that are written to run on more than one platform, or custom software written for more than one company often use `#ifdefs` to avoid redundant maintenance. If separate versions of a file are maintained for each of several platforms or customers, each version needs to be changed whenever an improvement is made to code that is common to all of them. `#ifdefs` are used to maintain different versions in a single file. Then, changes to common code only need to be made once.

If code becomes confused with many preprocessor directives, whatever time is saved on redundant maintenance could be lost just trying to figure out confusing source code. Hawk allows full benefit from the use of pre-processor conditionals by letting the code be seen more clearly for what is being worked on.

Because of the `Selective Display Mode`, the whole file is still included, but the parts that do not apply can be hidden.

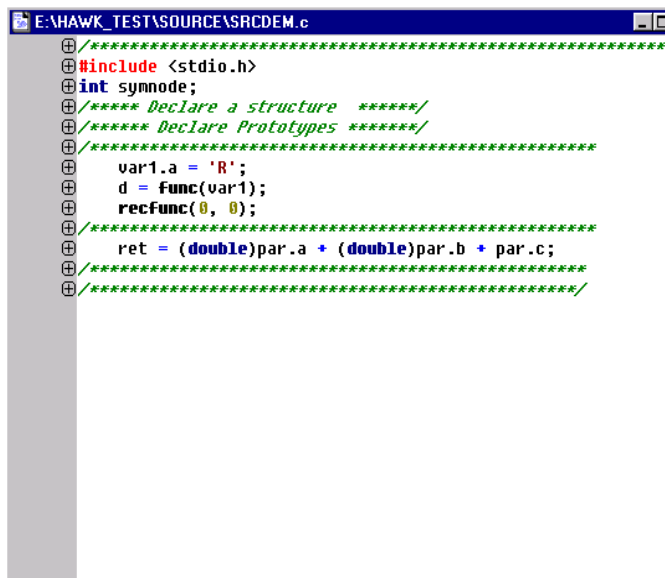
Adding "defines" to the `Preprocessor directives Defined Constants` field allows specified defines to remain visible in preprocessed view. Multiple "defines" can be specified by separating them with semi colons. Once the necessary "defines" have been added, and desired options have been chosen, clicking the dialog's `OK` button hides specified preprocess code.

Selective Mode

When a document is in `Selective` mode (shown in [Figure 10-5](#)), lines are preceded with a small button or icon. When the button contains a plus sign, it indicates that there is hidden text following that line.

- To view hidden text, double-click on the plus sign or press `SE` on that line. The plus sign then turns to a minus sign, to indicate that the section has been expanded to show the "invisible" lines.
- To collapse the section again, double-click on the minus sign or press `SE` again.
- Double-clicking with the right mouse button also expands and collapses hidden text.

Figure 10-5. Selective Mode



```

E:\HAWK_TEST\SOURCE\SRCD\EM.c
+ /*****
+ #include <stdio.h>
+ int symnode;
+ *****/
+ /****/
+ /****/
+ /****/
+ var1.a = 'R';
+ d = func(var1);
+ recfunc(0, 0);
+ /****/
+ ret = (double)par.a + (double)par.b + par.c;
+ /****/
+ /****/

```

Press **X** to restore the invisible lines. **CSc** restores *Selective Display* for the previously selected lines in the CUA or BRIEF keymaps. **#** restores *Selective Display* for the previously selected lines in the vi emulation.

Browsing, Tags and Symbols

Whether working on someone else's code, with a long line of inheritance, or working on code you wrote last week, a lot of time can be spent navigating copious amounts of unknown or forgotten code. Hawk provides the following text and source code tools for finding and navigating that code:

- [Browser](#)
- [Tags](#)
- [Outline Symbols](#)
- [Objects Window](#)

Each of the above tools have advantages and disadvantages when used for navigational purposes. The next few paragraphs describe reasons why one might choose one over another.

Using Navigational Tools

Browser

The best Hawk Browser uses the database file (symbols.udb) generated by the Ultra C/C++ compiler. The browser uses a graphical tree in a special tab of the Output window to demonstrate relationships and simplify locating necessary files or other pertinent information. See the Browser Support section below for more information on using this type of database.

Tags

Hawk Tags support provides only limited information about your code – primarily the location of function definitions. You can access this information in two ways: you can use the same graphical interface as the browser, or a hypertext-style lookup of the word at the cursor. This information, too, relies on you updating the database regularly.

Outline Symbols

Hawk's `Outline/Symbols` feature has its own graphical tree interface in the `Outline` tab of the `Project Window`. It also uses the `Symbols` tab on the `Output Window`, and background parsing to let you view the most up to date information about your project. Furthermore, `Symbols` support a large number of languages and can be readily modified to add to or expand the browse capabilities for new or existing languages supported by Hawk.

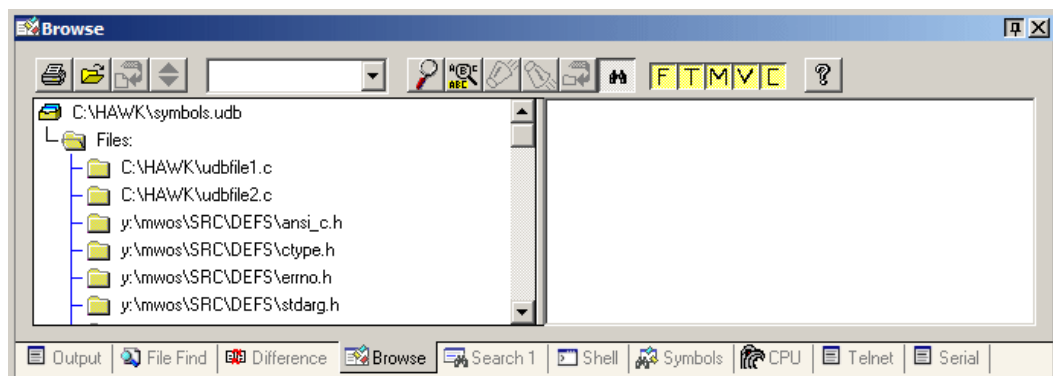
Objects Window

Hawk's `Objects Window` is a tab on the `Project Window`. It displays a hierarchical view of C/C++, C# and Java symbols that are associated with the name that is typed in the `Identifier box` at the top of the window. It can be used to view and browse code. Click on a symbol in the `Object Window` to access the position in the source file at which its corresponding code is located.

Browser Support

The Hawk `Browse` pane in the `Output Window` supports Ultra C/C++ databases. These are the databases produced by the Ultra C/C++ compiler. This kind of database is given the extension `.udb` by convention.

Figure 10-6. Browse Tab of the Output Window




The `Browser` is one of the windows available for viewing in the tabbed `Output Window`. It appears when you select the `Browse` tab on the `Output Window`. The `Browser` is actually two windows side by side with its own toolbar at the top. The `Tree` window appears on the left and the `Inspect` window appears on the right.

Press `?` in the `Browse` window to receive help, press `X` at any time to leave the window and return to the current edit window.dir

Selecting a Database

The first step in using the Browser is to select a database. If you have previously defined a valid `.udb`, `.bsc`, or `.ptg` file for the current project, that database is loaded when you select **Browse**. You define the database for the project in the **Directories** tab of the **Project Properties** dialog. If you have not defined a database, or if you later want to change databases, you may select a database by choosing the **File Open** button on the **Browser** toolbar.

Traversing the Tree








Once you have loaded the database, a tree structure will be displayed in the **Tree** window. You may traverse this tree from the keyboard by using the arrow keys to move from node to node, and by pressing the **Enter** key to expand or collapse a node. Using the mouse, you can double-click on a node to expand or collapse it, or click on the  (expand/collapse) button on the browser toolbar.

As you traverse the tree, note that information about the selected node appears in the **Inspect** window to the right. This window lists information such as the location of the item where it is referenced, if this information is in the database. You may move between the **Tree** window and the inspect window by either pressing the **Tab** key or clicking in the intended window.

Label Bitmaps

At the beginning of each line of the tree is a bitmapped label identifying to what the information in that branch relates. A key to these labels is given in [Table 10-4](#).

Table 10-4. Label Bitmaps

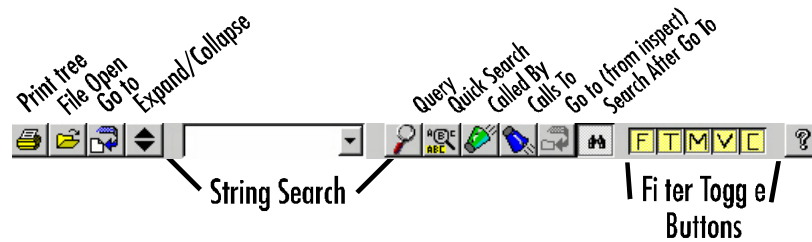
Bitmap	Description
	Root information coming from the database.
	Information pertaining to a file or module.
	Information pertaining to a class.
	Information pertaining to a function.
	A preprocessor macro definition.
	A type definition.
	Information pertaining to a variable.

The first file in the tree list can be labeled `<unknown>`. These are functions and other objects that are referenced, but whose source are not included in the current database.


Browser Toolbar


Refer to this diagram as you read the following descriptions:

Figure 10-7. The Browser Toolbar



Jump to Code

After traversing the tree, you will usually want to go to the corresponding source. There are two **Go to** buttons on the **Browser Toolbar** that allow you to do this. If you wish to go to the source code represented by the selected tree node (usually a point of definition), you can press the  (**Go to**) button on the left of the toolbar.

If you have selected a reference, a calling or called function in the **Inspect** window, you may wish to press the  key (**Go to from Inspect**) to go to that reference or function. Pressing the **g** key will always take you to whatever is selected in the **Tree** window.

String Search

The **String Search** feature  of the browser works like a **grep** or **filter** of information in the database. Use it to filter out extraneous information. It is one of the most useful features of the browser.

To use the **String Search** feature, enter a string into the combo box on the browser toolbar. Unlike some other browsers, the string you enter will not be treated in a case sensitive manner, and wildcards are supported. That is to say, if you enter "dump*", it will match strings like "Dump" and "dumpFile". Use the * character to match zero or more characters. Use the ? character to match any single character. Use the regular expression "set" syntax to match a single character from a set of characters. Set syntax supports character ranges in the form <start>-<end> and a single minus(-) to negate the matching set. For example, a*[cv] would match symbols like argc and argv. The pattern arg*[-cv] would match symbols that start with arg, but do not end in c or v.

Each matching identifier is displayed as a node on the tree. All other nodes disappear. You can then traverse this "filtered" tree.

Query Button


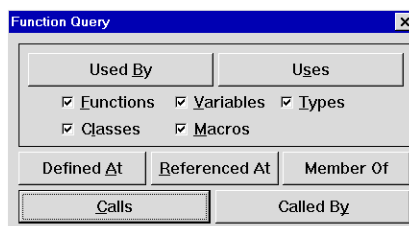
The first button to the right of the **Search String** combo box on the **Browser Toolbar** is the **Query** button. You can initiate a query by pressing the  button, or by pressing the **q** key. Depending on the context, one of six query dialogs will appear. An example dialog is shown below:

Figure 10-8. Tags Function Query




Each query dialog has a set of action buttons and filter check boxes. The filter check boxes apply only to the action buttons that are enclosed in the same group box. Therefore, in the example above, the check boxes apply only to the `Used By` and `Uses` buttons. The other action buttons operate independently of the filters.

With all the filters enabled (all the check boxes checked), you may find that you have too much information to sift through to find what you are after. It is probably best to check the minimum number of boxes that fill your needs.


Table 10-5. Query Dialogs

Button	Purpose
General	General inquiries about the terminals (leaves) on the tree.
Class	OOP class inquiries.
Function	Inquiries about functions.
Database	Inquiries from the root of the database.
Module	Inquiries related to a module.


Quick Search

The `Quick Search` button  on the `Browser Toolbar` is used when you are in a normal edit window and want to look up the word at cursor in the browser database. This step saving device acts the same as if you had typed the word into the `String Search` combo box on the `Browser Toolbar` and pressed `Enter`.

Called By/Calls To

When you have selected a function in the `Tree` window, you may view either a list of functions that call it, or a list of functions it calls by pressing one of the `Called By/Calls To` buttons . These functions do not work with `.ptg`; only with `.bsc` (Visual Studio) and `.udb` (Ultra C/C++) databases.

Search After Go To

The `Search After Go To` button  is a toggle that determines what action is taken after jumping to the related source code line (`Go To`). When this button is depressed, the `Browser` executes a forward search for the item selected in the `Browser` window from which it jumped. This is especially useful if you have modified

the source code since the last time the database was generated. The `Browser` may still be able to take you directly to the item of interest.

Filter Toggle Buttons

The `Filter Toggle` buttons allow you to focus in on specific categories of objects by turning filters on and off. Each of the lettered buttons at the right of the browser toolbar has an on and off position. When depressed, the button is on, and object associated with that button will show up in the tree. When the button is out, it is in its off position. Related objects will not show up in the tree. You can reverse the condition of any of these buttons by clicking on it with the mouse or by pressing the corresponding letter on your keyboard (such as `ftmvc`).

Below is a list of the buttons and their related objects:

Table 10-6. Buttons and Related Objects

Letter	Filter Objects
F	Function references and definitions.
T	Type references and definitions
M	Preprocessor macro references and definitions.
V	Variable references and definitions.
C	OOP Class references and definitions.

After changing the filter toggle settings, you will find that the new settings do not take effect until you have performed some action, such as opening a branch of the tree. To see the effect of the filter change on a branch you are already viewing, collapse and re-expand the branch.

Tags Support

Hawk has support for CTags and other Tags generating programs whose databases conform to the standard Tags format. The program `tagsw32.exe` is provided for generating Hawk-compatible tags, use the `-t`, `-oc`, and `-p` options to compile a tags database. The text file documentation for tags files is at `/MWOS/DOS/BIN/tags.doc`.

In addition, Hawk comes with a built-in Tags database generation capability. This program produces both a standard Tags database and a compiled database. The resulting compiled database may be used with Hawk's Browser in a similar fashion to a browser database. You can search the database and traverse its contents via a graphical tree.

A standard Tags database adds browse-like capabilities to an editor that knows how to read and use the database. If you see the name of a function on the screen and need to know what that function does, Hawk's tags functions allow you to jump directly to the original location in the original buffer.

Hawk is known to be compatible with the PCTags program from Modern Software and GNU Tags, the former of which is available for download from Starbase's FTP sight at:

```
ftp://ftp.premia.com/pub/es2vd108/addons/archive/
```

The latter is supplied with Hawk.

Tags Setup

To get Hawk to generate and use Tags databases automatically requires the following steps:


1. Create a project whose members are the files you want to scan for your Tags database. Refer to [Chapter 7 Projects, Project Spaces, and Workspaces](#) for information on creating a Hawk project.
2. Define which files hold your Tags data – your standard Tags database and your compiled database for use with the `Browser`. There are two entries on the `Directories` tab of the `Project Properties` dialog that store the name and location of these database files. The entries are listed as `Browser Database`, and `Tag Database`. If you are using Hawk's compiled tags utility, be sure to enter a filename in the `Browser` database field that employs the `.PTG` extension.
3. Select `Build Tags` from the `Project` menu.
4. Use the `Browser` or `TagFind` function.

Using the Tags Database

To use the standard Tags (not compiled) database, you need to have the function `TagFind` assigned to a key. The `TagFind` function looks at the word at the cursor and tries to find a match for it in the database. You can assign this function along with `TagNext` and `TagPrev` to keystrokes, if they are not already assigned. You do this through the `Keyboard` dialog on the `Tools -> Customize` menu.



For more information on the `TagFind` API, refer to Hawk's online help. For more information on custom keystrokes, refer to [Chapter 12 Customizing the Interface](#).

To use the compiled tags database with the `Browser`, load the `.PTG` file by pressing the `Browser File Open` button  and browsing for the file. You will then be able to traverse a tree of your tags database, search the database, and do queries for functions and other objects (barring friend classes). You will only be able to query definitions, however. References are not stored in the Tags database.



For more information about the `TagsWnn` utility provided with Hawk, see [Appendix A](#) at the end of this manual.

Outline Symbols: Overview

Outline Symbols make up a new type of browser for navigating and investigating code. They are named Outline Symbols because they use two system windows: one called the `Outline Window` (a tab on the `Project Window`) and the other called the `Symbols Window` (a tab on the `Output Window`). Outline Symbols constantly

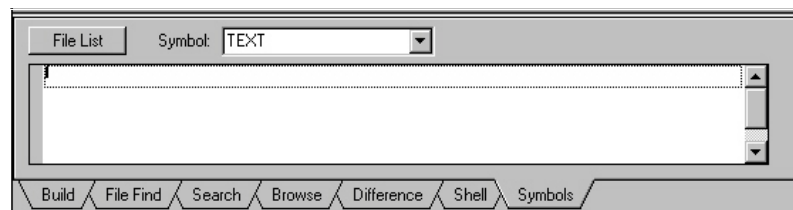
update in the background (there is no compiling required) and they have broad application, that is, they will work with virtually all languages for which Hawk provides support (C++, Java, Delphi/Pascal – even .INI files).

What are Symbols?


Symbols are any element of code for which a parser has been written. Parsers are regular expression patterns used to find and separate designated string patterns from code with the intention of displaying those string patterns in either the `Symbols Window`, the `Outline Window`, or both. Typically, symbols include functions, classes, macros and the like. New parsers can be added by creating the proper regular expression parser. Files are scanned using these parsers and the locations where symbols are defined are collected and placed in the symbols database.

Hawk's symbols parsing starts with the member list of the current project. Other files can then be added as part of the list to be scanned. The scan list can be modified in the `Edit Symbol File List` dialog, accessed by pressing the `File List` button in the `Symbols Window`. For example, you might want to scan MFC symbols, but not include the MFC source files in each project. To accomplish this, add the MFC symbols to the scan list instead.

Figure 10-9. Symbols Window



The `Symbols Window` is normally used in conjunction with the `Outline Window`, to which it has a special link. This link allows the `Outline Window` to pass symbol names to the `Symbol Window` for lookup, thereby automating the process of viewing symbols in the `Symbols Window`. Together, the `Outline` and `Symbols Windows` provide Browser-like capabilities for a variety of file types without any need to compile to keep current with code changes. The `Symbols Window` allows you to list the location or locations where a symbol is defined.

The `Symbol Box`  at the top of the `Symbol Window` normally reflects the symbol selected in the `Outline Window`. A history list is maintained in the drop-down list under the `Symbol Box`, recording the recent entries in this box. Also, when the cursor is positioned in the edit buffer, Hawk will pick up whatever the cursor is sitting on and display it in the `Symbol Box`. Then a background search is done through the symbols database to see if there are any other occurrences of this symbol.

If Hawk doesn't display anything in the `Symbols Window` it is likely that the symbol is not contained in the database. There are two reasons that a symbol may not be stored in the database:

- The symbols parser(s) (regular expression) does not match the string.

- It is not the Definition of the symbol (Hawk doesn't display References or other uses of the symbol).
- If the symbol being displayed is only located in one file, the associated code is displayed in the window. If there is more than one symbol in several locations, a list of files containing those locations will be displayed in the `SYMBOLS` window. The locations can be selected from the list by double clicking on a filename in order to open the file in the buffer editing area. The cursor will be positioned at the indicated location.

Symbol Scanning

The `SYMBOLS` window conducts a detect scan on listed files after a 3-second idle period for several reasons:

- To determine if any of the time/date stamps on the files in the database have changed.
- To determine if any newly created functions need to be added to the database.
- To determine if any newly added project files need to be added to the database.

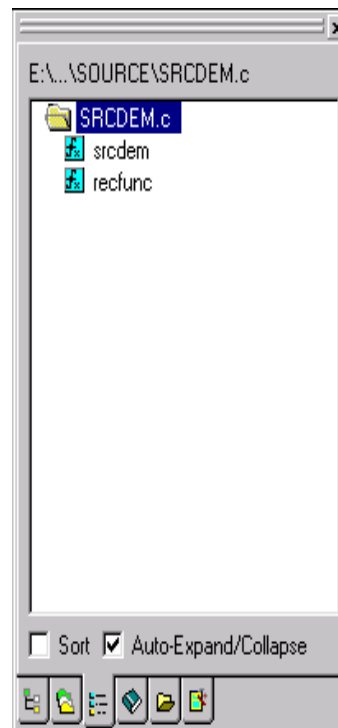
Only if it detects these changes, does it go ahead and start updating the database. If the symbols scan detects a keystroke, it will interrupt its detect scanning and begin again where it left off on the next scan. The symbols scanning and updating can be disabled (see [Symbols Database](#) below).

Outline Window

Hawk's `Outline` window displays symbols in a graphical tree, with each loaded file displayed as a root. As files are loaded, or edit windows are changed, the symbols found in that file are displayed beneath that file's name in the `Outline` window.

A single mouse-click on the symbol of interest displays information for that symbol in the `SYMBOLS` window. A double-click also moves the edit window to the defining location. When you single click on a symbol listed in the `Outline` window, as described above, the view in your current edit window is not modified. This allows you to browse symbol references without losing your place. If you double-click on a symbol listed under one of the loaded files, your view in the current edit window moves to that location, in addition to listing references in the `SYMBOL` window.

Figure 10-10. Outline Window



As you load files or change edit windows, the `Outline` window lists all open buffers as folders, and if the `Auto Expand/Collapse` check mark is checked (described below), automatically expands the view of the current buffer to show its matches. Double-clicking on any of the listed folder icons will always expand the outline-view, if there are symbols to be displayed for the file.

The symbols displayed in the `Outline` window are listed in the order in which they appear in the source file. They can also be displayed in alphabetical order, by clicking the `Sort` check box, as described below.

There are two check boxes at the bottom of this `Outline` window tab:

- `Auto-Expand/Collapse` turns off the automatic expansion of the symbols matches. When this check box is checked, changing from one document to another will close one list of symbols and open another for the newly current document. When the box is not checked, you control when the symbol lists open and close.
- `Sort` alphabetically sorts the list of symbols in the window.

Outline Scanning

The `Outline` window has a threaded background scanning process that runs after 1 second of idle time. It scans the opened source files to detect new symbols to be added to the view. The updating only occurs when changes have been made, but the detect scan is almost always running. Available parsers can be enabled in the `Outline Parsers` dialog, accessed by clicking the `Symbol Patterns` button in the `Customize -> Language -> CodeSense` dialog. A check box to indicate if

the symbol type should be displayed precedes each parser name. When you select a parser name, the settings in the dialog reflect those for that particular parser name. More marked check boxes, mean more displayed symbols (assuming the parsers are legitimate), and more background scanning.

If you have all the parser check boxes for a particular file type marked, you might experience a delay when opening or closing files because of the scan process. If you do, try un-checking the auto-expand box in the `Outline` window, or disabling some of the parser types.

Symbol Parsers

Symbols, as you recall, are any element of the code for which a parser has been written. Typically, these are Functions, Classes, Macros and the like. You can add parsers by creating the proper regular expression pattern, and inserting it appropriately in the designated dialog. `Symbol Parsers` are regular expression patterns used to parse particular code-elements covered by the current parser name. Patterns can be developed using the Hawk's Search mechanism. The online help in the `Outline Parsers` dialog provides information on each of the entries in this dialog, and provides some tips on creating Regular Expressions. Regular Expressions are also described previously in this chapter, under the topic [Regular Expressions](#). Files are scanned using these parsers.

The dialog for configuring parsers is accessed by clicking the `Symbols Patterns` button on the `Customize -> Language -> CodeSense` tab. Use the following steps to make a parser:

1. In the `CodeSense` dialog, highlight the file type of the file for which the parser is being made or viewed.
2. Click the `Symbol Patterns` button.
3. View the box that lists the parser names or types of symbols recognized by the current symbol parser. If the box is empty, there is no parser for the selected file type extension.

Example

If `.CPP` is selected as the file type, there should be three parser name entries in the `Parser Type` list box: `Function`, `Define`, and `Class`.

1. Click `Add`. The `Add Parser Name` dialog comes up.
2. Give the new parser a name and choose a type (such as `function`, `prototype`, `procedure`) from the `Type` drop-down list. The type determines what icon Hawk uses in the `Outline` window.
3. Click `OK`.
4. Back in the `Outline Parsers` dialog, highlight the new parser.
5. Type or paste an appropriate regular expression in the `Pattern` box. The string provides feedback, indicating whether the selected regular expression is `OK` or `Illegal`.

6. Choose or enter the characters that should be found by the parser in the `Keyword Characters` section.
7. Use the `Keyword Position` options to indicate whether characters to the left or right of the indicated cursor position (`\c` in the regular expression pattern) should be displayed in the `Outline` and `Symbols` Windows.
8. Click **OK**.

The new symbols should appear in the `Outline` Window when files of the appropriate file type are opened in Hawk.

Symbols Database

As mentioned, parsers used to match and display symbols are defined in the `Outline Parser` dialog accessed by clicking the **Symbol Patterns** button in the `Customize -> Language -> CodeSense` dialog. They are specific to the selected file type extension. Any symbols that are parsed are subsequently stored in the symbols database.

The symbols database (the file that stores gathered symbols) location is defined in the `Project -> Settings -> Directories` dialog. By default this database is located in the main Hawk directory. More than one symbols database can be used, by changing the name/location of the file. To maintain separate symbols databases for individual projects, put the full project path for each Symbols database in the `Symbol File` field of the `Directories` dialog.

There are two check boxes that apply to the symbols database file defined in the `Project -> Settings -> Directories` dialog. The check boxes are available when the `Symbol File` field is selected. The check boxes have the following functions:

- When the option `Auto-Update Symbols Database` is selected, Hawk automatically update the symbols database with any new functions when that file is saved.
- When the option `Show Definitions in Symbols Tab` is selected, automatic detection and scanning takes place for updating the `Outline/Symbols` Window and the symbols database.

Pop-Up Symbols Menu

Right-clicking on the **Symbols** tab of the `Output Window`, or right-clicking on the filename in the `Outline` tab of the `Project Window` gives a pop-up menu with three choices:

- `Rescan Outline Symbols` updates the information in the symbols database for the file currently selected in the `Outline` Window.
- `Rescan Symbols DB` deletes the database and rebuilds it. This is performed as a background task.
- `Compact Symbols DB` selection optimizes the symbol database for the current project by deleting unused records. This is performed as a background task.

Objects Window

The Objects Window is a tab on Hawk's Project Window. It is used to view and browse code. It displays a hierarchical view of C/C++ and Java objects/symbols that are associated with the name that is typed in the window's Identifier box. The objects in the hierarchy can consist of C/C++ and Java classes, functions, data, macros, types, etc. as they apply to the language being used. Each object-type has its own image, as displayed in the `Filter/Legend` dialog.



See "Filter/Legend Dialog" on page 229.



Display an Object Hierarchy

The Objects Window gets its information from CodeSense, so the same information sources that are used for CodeSense must be available for the Objects Window before it will work (see the topic *Objects Window and CodeSense* in this chapter). With that in mind, there are two ways to display an object hierarchy in the Objects Window: by typing an identifier in the Identifier box, or by using Hawk's standard popup menu. Those methods are described in the next topics.

- To control the quantity of hits returned in the Objects Window, adjust the Maximum Number of Hits slider on the CodeSense Global Configuration dialog.

Type an Identifier to Display an Object Hierarchy

To display an object hierarchy by typing in an identifier, do the following:

1. Type the name of an object/symbol in the Identifier box at the top of the window. Use the asterisk (*) wildcard character to match any characters to the right of the asterisk (e.g., `CMain*` will match all symbols that begin with 'CMain'). If necessary, click the Ignore Case button  to make the window case insensitive to the characters in the Identifier box (the button will appear depressed).
 - Typing only an asterisk (*) will result in a search for all symbols, but will be restricted to the project database. If you want to restrict a large search (e.g. `C*`) to the project database and open documents, click the Include CodeSense Library Databases button  to turn it off (button will no longer appear depressed).
1. Press `Enter`. A hierarchy will display.

Use Hawk's Right-Click Popup Menu to Display an Object Hierarchy

To display an object hierarchy by using Hawk's right-click popup menu, do the following:


1. Open a `.C`, `.CPP`, or `.JAVA` file.
2. Right-click on, or select and right-click on, a symbol (class name, function name, struct, etc.) in the open file. Depending on whether a selection was made, you will see the following in the resulting popup menu:

- If a selection was made you will have the choice to Browse for <selection> (exact match) or Browse for <selection>* (wildcard match).
- If no selection was made you will have the option to Browse for <symbol name> where <symbol name> is the name of the symbol that the cursor is on.
Click on any of the above options to display an object hierarchy.

Using the Objects Window

The Objects Window can be used to view and browse C/C++ and Java objects/symbols. Click on an object/symbol in the window to access the position in the source file at which its corresponding code is located.

Hover tips are available for each object/symbol in the hierarchy. The tip appears when the mouse cursor hovers over an object/symbol's representative icon. It provides additional information about the object/symbol in question.

Note that the Identifier box in the Objects Window will be case-sensitive if the `Ignore Case` button is off (button is not depressed). Click the button on  (appears depressed) to make the window case-insensitive to the string that is typed in the Identifier box.

Objects Window and CodeSense

The Objects Window gets its hierarchy information from CodeSense. Therefore, one or more of the following must be true before any information will display:

- One or more C/C++ and/or Java files must be open in Hawk.
- A project containing C/C++ and/or Java files must be open in Hawk.
- One or more legitimate CodeSense library databases must be listed and checked in the `CodeSense Global Configuration` dialog. If the correct library is not listed in the dialog, it can be added.
- One or more C/C++ and/or Java files must be listed in the `Symbols Window Edit Symbol File List` dialog.



See “CodeSense” on page 88.

Objects Window Popup Menu


If you right click on an object in the Objects Window you will get a popup menu with the following options:

- `Sort Alphabetically` - to sort the objects/symbols alphabetically.
- `Sort by Character Set` - to sort the objects/symbols by the ASCII values of the characters in the name and according to the code page currently in use. This allows the sort to respect locale-specific information (i.e. international characters).
- `Sort by Objects` - to sort by objects/symbol type.
- `Display by Definition` - to display objects/symbols by their definition. For example:

```
MsgPopupPromptHistory(LPXSTR histName, LPXSTR prompt, LPXSTR
response)
```

- `Display by Symbol name` - to display objects/symbols by name. For example `MsgPopupPromptHistory`
- `Filter/Legend` - to access the `Filter/Legend` popup menu.
- `Show Members in Root` - to show both class detail and function detail if class and function names are the same.
- `Ignore Case` - to toggle the `Ignore Case` setting on/off, telling `CodeSense` whether or not to consider case when lookups are performed.
- `Include CodeSense Library Databases` - to force the `Objects Window` to include information in `CodeSense Library` databases when displaying a hierarchy. If this is toggled off, searches are restricted to the project database and open documents, speeding up the process of displaying hierarchies.
- `CodeSense Editing Options` - to access `Customize -> Language -> CodeSense`.
- `CodeSense Global Configuration` - to access the `CodeSense Global Configuration` dialog.
- `Refresh Display` - to display the most current list of applicable objects/symbols (refreshes the current query, maintaining tree expansion).

Choose the item `Filter/Legend` on the `Objects` popup menu to access the `Filter/Legend` popup menu:

The `Filter/Legend` popup menu lists a set of object filters. Object filters control which objects/symbols will display in the `Objects Window`. Unmark any filter to prevent objects of that type from displaying. When filters are thus applied, the `Filter/Legend` dialog button  appears depressed.

Filter/Legend Dialog


One of the items on the `Filter/Legend` popup menu is `Filter/Legend`. It accesses the `Filter/Legend` dialog. The `Filter/Legend` dialog serves some of the same purpose as the `Filter/Legend` popup menu, providing the same list of `Objects` filters. The dialog also provides another set of filters labeled `Access`, that provide additional control over `Objects` filters in the following ways:

- By filtering objects according to public, private, and protected access attributes.
- By filtering objects that are non-members.

For example, if one wanted to see only public classes in the `Objects` window, they would mark the item `Classes` in the list of `Objects` filters, then mark the item `Public` in the list of `Access` filters.

When using `Objects` and `Access` filters in the `Filter/Legend` dialog, keep in mind the following:

- Marking or unmarking any of the `Objects` filters in the `Filter/Legend` dialog will mark or unmark the same filter on the `Filter/Legend` submenu.

- At least one `Access` filter must be marked in order for any marked `Objects` filters to display in the `Objects Window`.
- At least one `Objects` filter must be marked for any marked `Access` filters. Otherwise the `Access` filters are meaningless, and objects will not display.
- When all of the `Objects` and `Access` filters are not selected on the `Filter/Legend` dialog, filters have been applied and the `Filter/Legend` dialog button  appears depressed.
- Use the buttons `Select All` or `Clear All` to select or clear selections for all filters.

Symbols vs. Objects vs. Tags

Hawk offers four browsing features, `Symbols`, the `Objects Window`, `Tags`, and `Browse`, all of which have been discussed above. Each method offers its own advantages, some of which are listed below.

Pros and Cons of Tags, Browsers, Symbols, and the Objects Window

The following items describe some pros and cons of the different browsing tools available in Hawk:

- Hawk's compiled `Tags` databases (`.PTG` files), provide more limited code-information, but can be built from within Hawk.
- The `.PTG` files are displayed in the `Browse` tab of the `Output Window`. The files are displayed in a graphical tree format with mouse-click access.
- `Tags` and `Symbols` do not contain extended information about where symbols are referenced or called. They only display where the symbols are defined.
- `Symbols` do not have to be compiled. They are created in real-time, instantly displaying symbols as changes are made to edit-buffers.
- The `Objects Window` contains more extensive `C/C++` and `Java` information, as provided by the `CodeSense` library databases that are listed in `Customize -> CodeSense Global Configuration`.
- `Symbols` are displayed in graphical tree-format with mouse-click access to their respective file locations.
- `Symbols` are interactively customizable, offering the ability to display more than just the pre-defined elements in the `Outline Window`.
- `Symbols` are available for a wide variety of programming languages, and again, they are customizable so that more can be added for new languages if necessary.



Bookmarks

Hawk's bookmarks are handy for navigating code. Like any bookmark, they reserve current positions in documents so that it is easy to return to those positions when necessary. This section describes using the `Hawk Bookmarks` feature.

Global and Local Bookmarks

Bookmarks are used to facilitate movement from one position to another, within and between documents. There are both local and global bookmarks. Local bookmarks are known only within the document in which they were defined. Global bookmarks, on the other hand, are recognized from any document. Global and local bookmarks can be given names, as well as numbers, to help remember what the purpose of the bookmark was.

Graphical Bookmark Images

If `Show visible bookmarks` is turned on in `Customize -> Environment -> Bookmarks` (described below), graphical images representing bookmarks are placed in the left margin of the document in which the bookmarks were placed. There are two separate bitmaps: one for global bookmarks  and one for local bookmarks . The only difference between the two is the 'L' in the center of local bookmark bitmaps.

Setting and Removing Bookmarks

Bookmarks can be set and removed in a number of ways:

- The `Document` menu has a `Mark` option which sets a global mark at the cursor position. The graphical mark is placed in the left margin of the document.
- The various keymaps available in Hawk each have respective keystrokes for setting bookmarks. For example, in CUA, the keystroke for setting a global mark is `Ctrl-[0-9]`.
- Right-clicking the mouse at the far left margin of a document causes a pop-up menu to appear, with the following options, two of which are used for setting marks:
 - `Set a Global Mark`: Sets a global bookmark.
 - `Set a Local Mark`: Sets a local bookmark.
 - `Go to Bookmark Dialog`: Accesses the `Tools -> Customize -> Environment -> Bookmarks` dialog, described in the following section.

Bookmarks Dialog

Clicking the `Bookmarks` tab of the `Customize -> Environment` dialog accesses the `Bookmarks` dialog. The `Bookmarks` dialog allows you to set the visibility and naming attributes for local and global bookmarks. You can also view and delete bookmarks from the database in this dialog.

The desired attributes for both local and global bookmarks are selected in the `Bookmarks` dialog:

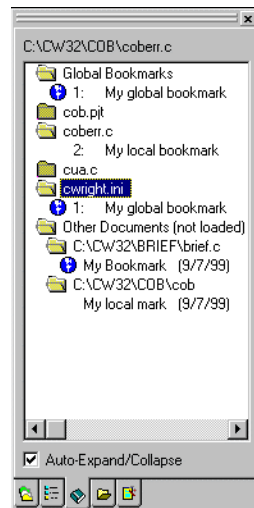
- When bookmarks are visible (when the `Show visible bookmarks` options are turned on), a marker appears in the left margin of each line that has a bookmark (as described in the section on [Graphical Bookmark Images](#)).

- When prompting is selected, a prompt for an optional bookmark name appears when a bookmark is dropped. This option only applies for bookmarks that are dropped using the applicable key combination for the current keymap.
- The options in the `Mark Name Database` section request that named marks be maintained in a database for future sessions and control how those saved names will be used in the future.
- The `Save Current Position` and `Restore Current Position` options save and restore the "current position" of documents when those documents are closed. These positions are maintained in the bookmark database. The "current position" entries in the database are never deleted unless `Delete entries older than threshold` is checked, in which case the current age specification applies.

Bookmarks Window

The `Bookmark Window` is a tab on the `Project Window`. It gives a view of local and global bookmarks defined in your documents. Clicking on a bookmark in the window moves you to the bookmark's document and location. The contents of the bookmarks database, and other bookmark options are found on the `Bookmarks` tab of the `Customize -> Environment` dialog.

Figure 10-11. Bookmark Window



Bookmarks Window: Global Bookmarks

Global bookmarks are listed `Bookmarks Window` under the `Global Bookmarks` heading, regardless of the document they are located in. The entry for the global bookmark shows:

- the bookmark number
- the bookmark name--if one was given
- the name of the document in which the bookmark is found

Bookmarks Window: Local Bookmarks

Local bookmarks are listed in the `Bookmarks Window` under the name of the document in which they are found. The entry shows the following:

- the bookmark number
- the bookmark name—if one was given

Double-clicking the document heading makes that document current.

Bookmarks Window: "Other Documents" Node

The `Other Documents (not loaded)` heading at the bottom of the `Bookmark Window` lists files that are not loaded but have entries in the bookmark database. Each file under the `Other Document` heading lists individual named marks associated with the file and the date stamp of the mark. Double-clicking on one of the files loads it. The file will be grayed if it does not exist (and the double-click action described will not occur).

Auto-Expand/Collapse

When the `Auto-Expand/Collapse` check box at the bottom of the `Bookmark Window` is checked, changing from one document to another will automatically close one list of bookmarks and open another for the newly current document. This saves a step when you want to view a list of bookmarks. When the box is not checked, you control when the bookmark lists open and close.

Button Links

Button Links are special action buttons that Hawk lets you embed in your text files. You may use them to view bitmapped images, bring up related documents or spreadsheets, run macros or just to make notes. To any other editor, it is still just a straight text file, and because the buttons are placed in comments, source code files compile as they normally would. Button Links are a navigational tool in that they maintain the current position of the file while the link goes to a different position or file.

How it Works

You select the type of link and the text that will appear in the button. Hawk uses this information to create an index entry for that text, and places it into the text file with a special 3 character prefix and suffix. Comment prefixes and suffixes are used as you indicate. The index entry refers to an entry in a database that indicates what the button link does.

What you See

When you have turned on the `View Links` option in `Customize -> View Setups`, you see a 3D-style button containing the text you specified, and a notation indicating what type of link it represents. Click on the button and the associated action is performed.

Defining Buttons

Button Links are defined in the `Insert Link` dialog, which is accessed by clicking `Insert Link` on the `Edit` menu. Enter your button's text (which must be unique), select the link type, and enter the appropriate text to be associated with the button. The nature of the associated text depends on which link type you select.

You can perform four different categories of actions using buttons:

- `Macro` – allows you to interactively execute any API command that is available interactively. The text associated with a Macro link is the function call. It is limited to a single function.
- `Templates` – allow you to insert text, prompt, move the cursor, or execute a series of actions. For templates, the associated text is the contents of a template to execute, such as those supplied to the `ExtExpandTemplate` function. (For more information about Hawk templates, see [Chapter 6 Editing Features](#).)
- `Document` – allows you to run the program associated with a file type, to view or modify that document. The document might be a bitmapped diagram or a word processor file. In this case, the text associated with the button is just the filename of the document.
- `Application` – lets you define any arbitrary command line to be executed when the button is pressed.
- `URL Button Links` – allows you to jump to a specified Web Address (URL) by issuing a shell command. The Web Page comes up in your default browser. This type of link could also be used for other shell commands.
- `Popup Note` links come in two flavors: standard notes and "to do" notes. These two types of notes are essentially the same. Providing two link types for notes, however, gives the buttons a different appearance and allows you to sort "to do" notes separately from other notes, when viewing the list of defined links.

11

Checking and Formatting Files

This chapter describes the Hawk formatting and file-checking tools. It includes the following sections:

[Differencing](#)

[Merging](#)

[Format Source](#)

Differencing

Hawk has a fairly sophisticated differencing feature that can be used directly from within the editor; it is not an external application. The Hawk differencing feature offers several advantages over other differencing utilities:

- The editing features on the `Side-by-Side Difference` window offer convenient and easy ways to selectively merge and edit two similar documents.
- The `Difference Controls` allow for easy navigation between differences.
- Differences can be recomputed after changes are made, and the documents being differenced can be immediately accessed at any point that direct editing is necessary.

The Hawk differencing feature can be used either side-by-side or interleaved. The following sections describe these two differencing methods.

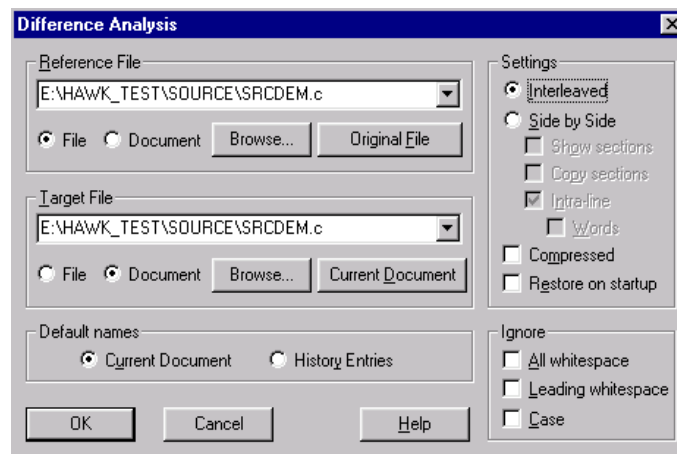
Interleaved Differencing

The Hawk Interleaved comparison merges two documents and places them into a new, separate document that can be edited and saved apart from the original documents.

Difference Analysis Dialog for Interleaved Differencing

All differencing in Hawk can be initiated from the `File -> Difference` dialog. To do an interleaved difference, choose the `Interleaved` option.

Figure 11-1. Difference Analysis Dialog (Interleaved)



- When the `Interleaved` option is marked, the `Show Sections`, `Copy Sections`, `Intraline`, and `Words` options in the dialog will become disabled. Those options are specifically related to `Side-by-Side Differencing`.

To use interleaved differencing, complete the following tasks:

1. Set the desired differencing options. The `Compressed`, `Ignore White space`, and `Ignore case` options are available, and provide the following functionality:

- `Compressed` displays only the parts of the files that are different.
 - `Ignore: All White space` causes all differences in white space to be ignored.
 - `Ignore: Leading White space` ignores differences in white space at the beginning of all lines.
 - `Ignore: Case` disregards any differences in case between the documents.
2. Mark the `Restore on Startup` option to tell Hawk to perform the current difference operation the next time Hawk is started. This option saves a step for on-going differencing projects that may take more than one day.
 3. Select from the `File` and `Document` radio buttons to determine what files will be displayed when the `Browse` button is pressed (The `Browse` button is used to browse for files to be compared.)
 - If the `File` radio button is on, the files displayed will be files that reside anywhere on your system.
 - If the `Document` radio button is on, the files displayed will be those that are loaded in Hawk.
 - Choose the files to be compared. Two files are needed when doing a difference, a `Reference` and a `Target` file.
 - The `Reference` file is normally used as the basis of comparison. It is usually the older of the two files. If comparing the current file with the same file on disk, just press the `Original File` button and its name will be inserted into the `Reference` field.
 - The `Target` file is the file to which the `Reference` file is compared. It is usually the newer of the two files being compared. If comparing the current document with the reference, just press the `Current Document` button, and the appropriate filename will be inserted in the `Target` field.
 4. The `Default Names` section of the `File -> Difference` dialog offers two features that help reduce the number of steps needed to browse for the files that will be differenced. Choose from the following:
 - When the `Current Document` radio button is selected, the files displayed in the `Reference` and `Target` fields will default to the document that is currently loaded in Hawk.
 - When `History Entries` is chosen, the files in the `Reference` and `Target` fields will default to the last files chosen when last in the `File -> Difference` dialog.
 5. Press `OK` to initiate the differencing. The differences will be displayed in a separate document in Hawk.

Interleaved Document

The title displayed in the interleaved document will be a combination of the names of the `Target` and `Reference` files.

Example

If two files, `foo1.c` and `foo2.c`, are being compared using the Interleaved option, the title of the resulting interleaved window would look something like the following:
`Foo1.c->Foo2.c`

The resulting interleaved document contains lines from both documents:

- The lines that are the same in both documents will have normal coloring in the interleaved document.
- Lines that have been added to the Target file will be preceded by plus signs (+). Lines that have been deleted will be preceded by minus signs (-). The added and deleted lines will also be displayed in different colors from each other and the rest of the text.
- An exclamation point (!) will precede those lines that differ in white space only, if options to ignore white space have been chosen.

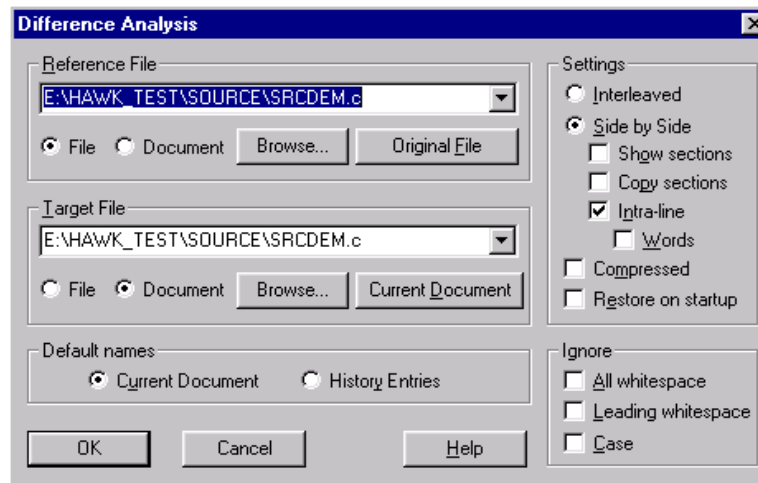
Side-by-Side Differencing

The Side-by-Side difference is shown in a separate tab of the Hawk Output Window (labeled `Difference`). Limited editing is allowed in the side-by-side difference window, with utilities provided for navigating the differences shown.

Difference Analysis Dialog for Side-by-Side Differencing

To compare two files using Hawk side-by-side differencing, go to the `File -> Difference` dialog and mark the `Side-by-Side` radio button.

Figure 11-2. Difference Analysis Dialog (Side-by-Side)



To perform a side-by-side differencing, complete the following steps:

1. When the side-by-side option is marked, the following four additional options are available, which were not available for Interleaved differencing: Show Sections, Copy Sections, Intra-line, and Words. These options are specifically related to the side-by-side Difference window. Mark any desired options according to the descriptions provided:

- Mark the **Show Sections** option to cause differing **sections** of the two files being compared to be completely highlighted in colors different from the rest of the text. If **Show Sections** is not marked, only the left margins of the portions that are different will be highlighted.
 - Mark the **Copy Sections** option to control how lines can be edited on either side of the difference window. If it is marked, whole sections can be deleted or copied from one side to the other. If it is not marked, only one line at a time can be copied.
 - Mark the **Intraline** option to show the **parts** of the lines that are different, rather than just flagging the whole line as being different.
 - Mark the **Words** option to keep the **Intraline** option from displaying every single character that differs in either document. It limits the display to show only the whole **words** that differ.
2. Complete the **Compressed**, **Ignore** and **Restore on Startup** options in the same way as for an **Interleaved** difference operation (refer to the previous topic, **Interleaved Differencing**).
 3. Choose the files to be compared for a **Side-by-Side** difference using the same process used for **Interleaved** differencing. The **Browse** buttons are controlled by the **File** and **Document** radio buttons, and the default **Reference** and **Target** file names are controlled by **Current Document** and **History Entries** radio buttons in the **Default Names** section of the dialog.
 4. When you have set the **File -> Difference** dialog with the desired files and settings, click **OK** to display the files, side by side, in the **Difference** tab of Hawk's **Output Window**.

Side-By-Side Difference Window

The **side-by-side** difference window shares common vertical and horizontal scrollbars to facilitate synchronized scrolling between the two documents.

Note the following characteristics of the **side-by-side** comparison:

- The **side-by-side Difference** window uses colors in the left margin to mark the points where the two files differ (If the **Show Sections** option has been marked, the differing text-sections will also be highlighted).
 - With the standard color scheme, inserted lines will be marked with red, modified lines are marked with blue, and deleted lines are marked with black in the left margin.
 - When intraline differencing is turned on, you will also see the differences within the line marked as blue or modified.
 - Lines that are common to both documents receive normal text colors.
- You will see these three patterns of text in the two windows:
 - Normal text in both windows
 - Text in one window, blank line in the other

- Marked text in both windows

View Mode

When you are presented with the *side-by-side* difference window, it is initially in "view only" mode. This mode allows you to see the differences but not to change the contents of either document. You may go to a location in either document and edit the file in the normal editing window, but this will not be reflected in the differencing window. The window can be changed to "editing" mode by clicking on the center-most button in the *Side-by-Side Difference* window's toolbar button.

Figure 11-3. View Mode Toolbar



Some of the buttons on the "view mode" toolbar are described below (in order, from left to right).

- The `Question Mark` brings up `Help` for the *Side-by-Side* difference window.
- The `Recompute Differences` button prompts to save the files contained in the difference window, saves them if desired, then recomputes the differences between the two files with the latest changes.
- The `Go To File` button loads the file in which the cursor sits in a Hawk buffer and moves the cursor to the point that it currently resides in the side-by-side difference window, preparing the file for direct editing.
- The `Change to Edit Mode` button changes the *Side-by-Side Difference* window to editing mode. The side-by-side edit mode is described in the next section.
- The `Difference Control` (triangular arrow) buttons are described in the topic entitled `Difference Controls`.

Edit Mode

Side-by-Side differencing allows limited editing to be done directly in the difference window. The editing features available in the *side-by-side* difference window are convenient for quickly moving lines and sections of text within and between the two difference windows. In order to maintain the integrity of the *side-by-side* difference, direct editing (that is typing) in the difference window is not allowed. The difference window initially displays in the `view mode`, as described above. The center-most (`Change to Edit Mode`) button on the difference window's toolbar toggles the difference window to `edit mode`, and back.

Figure 11-4. Edit Mode Toolbar



The `edit mode` buttons are described below (in order, from left to right).

- The `Question Mark` button brings up `Help` for the `Side-by-Side Difference` window.
- The `Copy Changes to Document` button copies the edits in the current difference window to the corresponding document.
- The `Save Changes` button saves the changes made to the current compared document.
- The `Difference Control` (triangular arrow) buttons are described in the topic entitled `Difference Controls`.
- The `Change to View Mode` button changes the window to `view mode`, as described in the [View Mode](#) section in this chapter.
- The `Copy selected lines to left` button copies inserted lines or sections from the right pane of the difference window to "deleted" (empty) or modified lines on the left. If no lines are selected, the current line is copied. If the `Copy Sections` option was selected when the difference operation was invoked, the current section is copied. After the copy, the selected lines are part of the text common to both files. If the lines are already part of the text common to both, the operation fails.
- The `Copy selected lines to right` button copies inserted lines or sections from the left pane of the difference window to "deleted" (empty) or modified lines on the right. If no lines are selected, the current line is copied. If the `Copy Sections` option was selected when the difference operation was invoked, the current section is copied. After the copy, the selected lines are part of the text common to both files. If the lines are already part of the text common to both, the operation fails.
- The `Delete Selected Lines` button deletes the selected lines in a window. The deleted lines are not removed, but rather are marked "deleted" and left empty. You may wish to do this in preparation to moving lines in the adjacent window. Add as many deleted lines as you like. They will not show up in the saved file.
- The `New Deleted Line` button inserts newly "deleted" blank lines in both windows. You may wish to do this in preparation to shifting lines up or down. Once again, add as many deleted lines as necessary. They will not show up in the saved file.
- The `Move Selected Lines Up` button moves the selected line up one line. If no lines are selected, the current line is moved. To use this button, there must be a deleted line above, or the line shifted must be a deleted line.
- The `Move Selected Lines Down` button moves the selected line down one line. If no lines are selected, the current line is moved. To use this button, there must be a deleted line below, or the line shifted must be a deleted line.

Difference Controls

The `Difference Controls` are displayed on the top of the `Side-by-Side` difference window as four triangular black arrows that point to the left and right. They are used to view the various areas of difference between the two documents.

- Press the far right arrow (the one with the vertical bar directly to the right of it) to jump to the last difference in the two files.
- Press the far-left arrow (also with a vertical bar) to jump to the first difference.
- The left and right arrows in between go to the immediate next and previous differences, if there are any.

toggling between the Two Panes of the Difference Window

There are three key-sequences for moving back and forth between the two *Side-by-Side* difference windows:

- `CSn, CWDiffNextWindow 1`: moves to right window (next)
- `CSp, CWDiffNextWindow 0`: moves to left window (previous)
- `CSo, CWDiffNextWindow 3`: moves to other window (next & wrap)

You can customize key-sequences for the difference window using the function `CWDiffKmapAssign` in the `Tools -> Customize -> Keyboard` dialog or the `[KmapAssign]` section of `MWHAWK.INI`.



For more information about Hawk's `CWDiff` APIs, consult the online help.

Side-by-Side Difference Printing

A `Print` button is available from the `View Mode` of the *Side-by-Side* difference window when both sides of the window are in the same mode (i.e. normal, compressed), and neither is in hex mode.

A modified `Print` dialog entitled `Print - Side by Side Difference` displays. Printing the contents of the *Side-by-Side* difference window is similar to printing a regular document in two-up mode, except that the left side will contain the Reference document/file, while the right side will contain the Target document/file.

Differences between the documents are shown as follows:

- Lines that are "deleted" in one side will display as a minus sign only.
- Lines that are "added" on one side will print, preceded by a plus sign.
- Lines that are modified will be preceded by an asterisk.

Directory Differencing

Directory differencing (also referred to as "tree differencing") allows you to specify two directories for comparison. Use the `Directory Difference Analysis` dialog to request directory differencing. To access the dialog, select `File -> Difference Directories`, or **right-click on the `Difference` tab of the Output Window and select `Difference Directories`**.

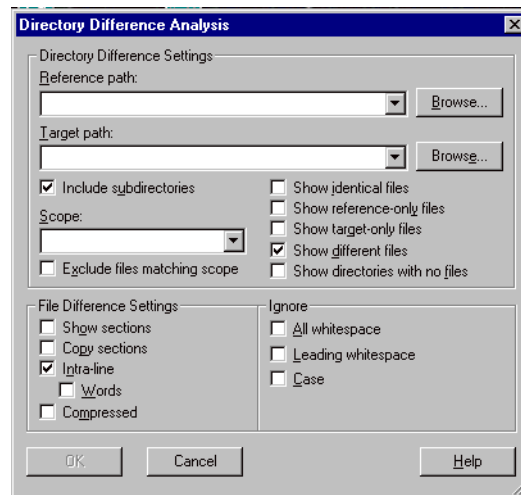
Hawk takes each file in the `Reference` directory and searches for the same file in the `Target` directory. Check the box `Include Subdirectories` to include all subdirectories of the `Reference Path` and `Target Path` in the directory comparison.

Results from the `Directory Difference Analysis` dialog are displayed on the `Tree Difference` window (found on the `Difference` tab of the `Output Window`). You can also run a `Directory Difference Report` that describes the results. Refer to the following sections.

Directory Difference Analysis Dialog






The `Directory Difference Analysis` dialog appears as follows:

Figure 11-5. Directory Difference Analysis Dialog



The settings in the `File Difference Settings` and `Ignore` groups on the `Directory Difference Analysis` dialog pertain to the side-by-side file differencing that is performed on a specific file pair when you double-click on the corresponding node in the `Tree Difference` window. The settings in the `Directory Difference Settings` group, described next, determine which files and directories are included in the tree directory in the `Tree Difference` window.

- **Reference Path:** Enter the path of the first directory of files to be compared, or press `Browse` to look for a directory on the `Select the Reference Directory` dialog. Typically the `Reference` directory will contain older, unchanged versions of a set of files.
- **Target Path:** Enter the path of the second directory of files to be compared, or press `Browse` to look for a directory on the `Select the Target Directory` dialog. Typically the `Target` directory may contain newer, edited versions of a set of files.
- **Include Subdirectories:** Check this box to include all subdirectories of the `Reference Path` and `Target Path` in the directory comparison.
- **Scope:** In the `Scope` field, enter a semicolon-delimited list of wildcard patterns to define which files are included in the directory differencing. If no `Scope` is entered, all files are included.

- Check the option `Exclude files matching scope` to exclude files matching the `Scope` from directory differencing instead. If the option is unchecked, files matching the `Scope` will be included.
- `Show Identical Files`: Check the option `Show Identical Files` to include files on the `Tree Difference` window that are the same in both directories. Filenames of identical files will be gray and preceded by the icon . Since the files are the same, you cannot double-click on identical files to difference them.
- `Show Reference-only Files`: Check the `Show Reference-Only Files` option to include files on the `Tree Difference Window` that only exist in the `Reference` directory. Reference-only files display in boldface, preceded by the icon . Since there is no file in the `Target` directory to compare it to, the file cannot be differenced.
- `Show Target-only Files`: Check the `Show Target-Only Files` option to include files on the `Tree Difference Window` that only exist in the `Target` directory. Target-only files will display in boldface, preceded by the icon . Since there is no file in the `Reference` directory to compare it to, the file cannot be differenced.
- `Show Different Files`: Check the `Show Different Files` option to include files on the `Tree Difference` window whose `Reference` and `Target` directory versions are different. Files with differences are shown in boldface, and preceded by one of the following icons:
 -  The Reference File (on the left) has been revised more recently than the Target File.
 -  The Target File (on the right) has been revised more recently than the Reference File.
- `Show Directories with No Files`: Check the `Show Directories with No Files` option to display subdirectories of the `Reference` or `Target` directory on the `Tree Difference Window`, even if the subdirectory contains no files. This option is disabled if you have elected not to `Include Subdirectories` in the directory comparison.

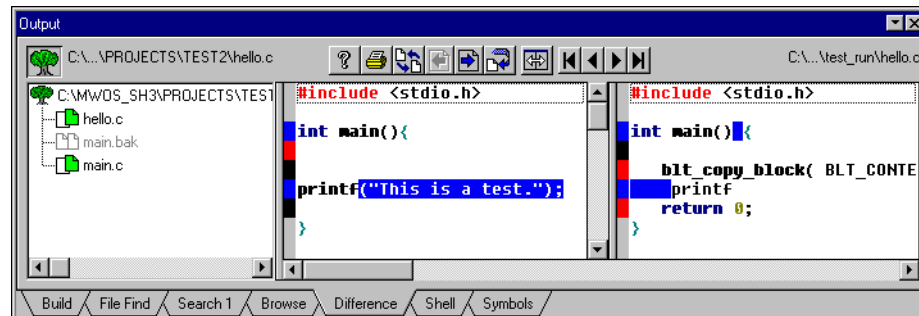
Tree Difference Window

The `Tree Difference` window shows the results of comparing the files in two entire directories, as requested on the `Directory Difference Analysis` dialog. A tree directory that integrates the `Reference` and `Target` directories is provided; click on any file in the tree that exists in both directories to show the specific differences in that file in `Side-by-Side Differencing` format.


Right-click on any file in the tree to see a file-level context menu. Select the menu option `Copy File` to copy the selected file from one directory to another on the `Copy File` dialog.



A sample directory differencing in the `Tree Difference` window is shown in [Figure 11-6](#).

Figure 11-6. Tree Difference Window (Difference Tab of Output Window)



Tree Differencing Toolbar










Press  on the `Tree Difference` window to show or hide the directory tree you generated on the `Directory Difference Analysis` dialog. When the directory tree is hidden (the button appears raised), the files currently being compared will expand to fill the `Tree Difference` window, revealing more of each file.




Use the `Previous Changed File`  or `Next Changed File`  button to display a `Side-by-Side Difference` on the previous/next file with differences in the directory tree.

All other toolbar buttons are the same as used for `Side-by-Side Differencing`.

Tree Directory Icons

The various file icons in the tree directory are defined as follows:

-  This small tree image indicates the root node. The text shows the `Reference` and `Target` directories used in the directory difference. Right-click on this tree icon to display a directory-level context menu.
-  or  The subdirectory exists in both the `Reference` and `Target` directories.
-  or  The subdirectory exists only in the `Reference` directory.
-  or  The subdirectory exists only in the `Target` directory.
-  The files are identical.
-  The file exists in both directories, with the file in the `Reference` directory being newer. Double-click on the item, or highlight and press `Enter`, to run a `Side-by-Side Difference` on the file pair.

-  The file exists in both directories, with the file in the Target directory being newer. Double-click on the item, or highlight and press **Enter**, to run a Side-by-Side Difference on the file pair.
-  The file exists only in the Reference directory.
-  The file exists only in the Target directory.

Directory Difference Report

The Directory Difference report shows analysis settings, file and directory counts, and a list of files in the directory tree. To allow user customization, the API Macro `DiffDirReport` is used to create this report. You may modify `DiffDirReport` by editing the macro on the **Tools -> API Macros** dialog.

A sample report is provided.

Figure 11-7. Directory Difference Report Sample

Report on Directory Differences

```
Reference Directory:  C:\CWM\dir1\
Target Directory:   C:\CWM\dir2\
Scope: *.*
Exclude files matching scope:NO
```

```
Count TypeFilesDirectories
-----
```

```
Different, newer reference9
Different, newer target2
Reference-only31
Target-only11
Identical23
Total 17 4
```

```
File Filters:
```

```
Show Identical Files:NO
Show Reference-Only Files:YES
Show Target-Only Files:YES
Show Different Files:YES
```

```
File List
-----
```

```
Target-only.\onlyN2\temp.txt
Different, newer target.\sub\subin\TEMP.C
Reference-only.\sub\subin\TEMP.H
Target-only.\sub\subin\TEMP.HTM
Different, newer target.\sub\TEMP.C
Different, newer reference.\sub\TEMP.H
Different, newer reference.\TEMP.C
Different, newer target.\TEMP.TXT
```

Using Difference Utilities

Hawk has several difference APIs that allow the user to access the difference features without using the mouse to go to the `File -> Difference` dialog. A list of these functions can be found by doing a search for the keyword `Differencing` in Hawk's `Help -> Search For Help On` dialog. These functions are available for use from Hawk's `Tools -> API Command` key, for binding to custom keystrokes, buttons or menu items, or to use in custom Hawk DLLs or macros.

Merging

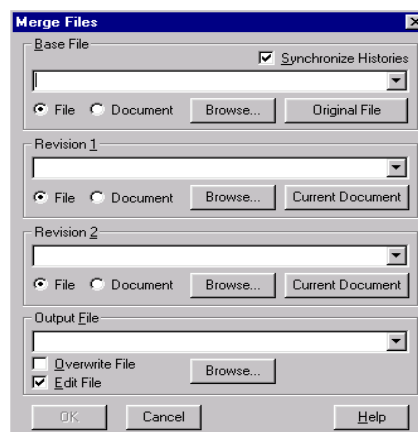
The `Merge Files` Dialog allows you to specify three files for comparison, and the name of the file that is to contain the merged changes. By merging files you can determine if there are conflicts between changes made to two or more versions of a file.

Using the Merge Files Dialog

To use the `Merge Files` dialog complete the following:

1. Select the `File` menu.
2. Choose `Merge`. The `Merge Files` dialog displays.

Figure 11-8. Merge Files Dialog



3. Complete the following fields:
 - `Base File`: The first filename you must enter is the name of the `Base File`. This is the file from which the two revisions were derived. This is usually the oldest of the three files.

`Revision 1` and `Revision 2`: These files are both revisions of the `Base File`. It does not matter which revision you specify as `Revision 1` and which you specify as `Revision 2`; the outcome of the merge is not materially affected. You may, however, wish to establish a convention for your

own use that `Revision 1` is "mine" and `Revision 2` is "theirs", or that `Revision 1` is older than `Revision 2`.



For the Base File, Revision 1 and Revision 2 fields:

- You may type in the name of the file, or select the **Browse** button to locate the desired file on disk or select from the document list.
- You may place the name of the file, from which the current document was read, into this field by pressing the **Original File** button (for the **Base File**) or the **Current Document** button (for the **Revision** files).
- The **File** and **Document** radio buttons allow you to specify whether Hawk should look for this file on disk, or whether it has already been loaded into a document.
- **Output File**: In this box, specify the name of the file to contain the output from the Merge. You may type in the name of the file or select the **Browse** button to locate a file on disk. This merge file is always written to disk, rather than just being placed into a document as the Differencing function does. (Refer to the topic **Differencing**, in this chapter.)
- **Overwrite File**: Check this box to tell Hawk to overwrite a like-named file when creating the output file.

If you do not check the **Overwrite File** option, and a like-named file exists, Hawk will prompt you for permission to overwrite the file. If you do not give this permission, Hawk aborts the merge.

- **Edit File**: Check this box to load the output file for viewing and editing at the end of the operation.
- **Synchronize Histories**: Select this option to have Hawk try to anticipate selections from the history lists. When you select the **Base** file from the history list, it automatically selects the **Revision 1** and **Revision 2** files that you used with that **Base** file last time.

Merge Output

The **Merge** function will alert you if it found any conflicts in the two sets of changes it is merging. If no conflicts were reported, editing the file is probably not necessary. When you do have one or more conflicts, you will need to know how to find them so that you can resolve the conflict.

Merge uses a series of ten dashes to separate the conflict area from the rest of the file. This enables you to search for conflicts in your file using the regular expression "`^-----`". This pattern will match ten dashes at the beginning of a line. Each line containing a message from the Merge function will begin in this manner. You will need to delete these lines after resolving the conflict.

The **Merge** function provides the following information:

- The first message from **Merge** identifies the conflict by number, names the **Base File**, and indicates that the **Base File** was read from disk (file).

- The next message introduces the section of the `Revision 1` file that conflicts with `Revision 2`. It also gives the filename and indicates the origin of the file (File or Document).
- The line or lines between this message and the next message from Merge are taken from the `Revision 1` file.
- After the lines from `Revision 1` there is a message from the Merge function introducing the corresponding section from `Revision 2`. Again Merge indicates the filename and origin.
- Between this message and the "End Conflict" message are the lines that you need to compare with those from the `Revision 1` file to see how to resolve the conflict.

On occasion, you may find that the section of conflicting line from one revision or the other contains zero lines. This is an indication that the lines were deleted in one revision and changed in the other. The revision from which the lines were deleted will be represented by an empty conflict section.

Once you have resolved the conflict and deleted the message lines you are ready to search for any further conflicts.

Removing Changes with Merge

Merge can also be useful when your development has been a linear progression, rather than simultaneous revisions (branching).

Example

You may find that a bug was introduced at some point in the past, and you want to undo those changes without abandoning the changes that you have made since.

You could do this manually, but Merge is faster and easier.

Usually, the `Base File` is the oldest of the three files used in the Merge comparison. It need not be the oldest, however. For the purposes of removing changes, the `Base File` is a revision in the middle that looks backward at the changes that must be removed and looks forward to the changes that must be retained:

1. When selecting the `Base File`, select the first revision following the changes you want to remove.
2. Next select the most recent revision of your file as `Revision 1`.
3. For `Revision 2`, choose the revision just before the change you want to remove was introduced.

Compared to the `Base File`, the Merge function will see one set of changes that removes the bug, and another that adds other changes. Since the `Base File` and the `Revision 1` file both contain the bug, the file without the bug (`Revision 2`) will look like a change made to a single revision. As such, the bug fix will quite likely be automatically incorporated into the resulting file.

Format Source

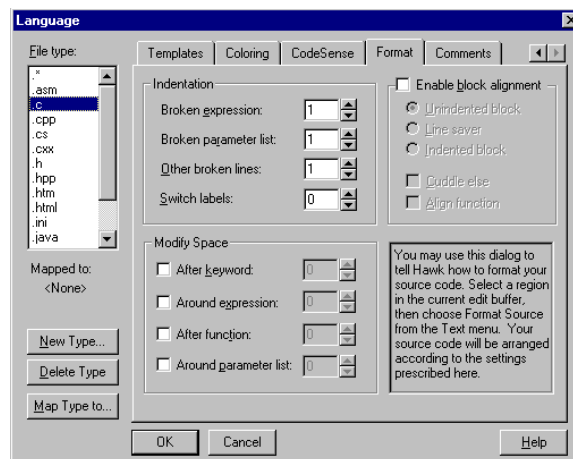
For your C, C++ and Java source files, Hawk provides functionality called `Format Source`. This allows you to select a block of code, and reformat it based on a pre-defined set of criteria (this is also known as beautifying the code).

Setting up Your Formatting Criteria

Before you can use the formatting capabilities within Hawk, you must establish certain criteria for Hawk to use. This is done on the `Format` tab of the `Language` dialog. To access this dialog:

1. Select the `Customize` menu.
2. Choose `Language`.
3. Highlight `.C`, `.CPP`, or `.JAVA` in the `File type` box. The criteria you set will only apply to the file type you have chosen here.
4. Click on the `Format` tab. The following dialog displays:

Figure 11-9. Customize -> Language -> Format Dialog



5. Complete fields in the following groups:
 - `Indentation`: The settings in this group modify how many tab stops to indent each item:
 - `Broken expression`—When an expression spans more than one line, indicate how much additional indentation to give to the continuation line or lines.
 - `Broken parameter list`—When a parameter list spans more than one line, indicate how much additional indentation to give to the continuation line or lines.
 - `Other broken lines`—When a statement other than an expression or parameter list spans more than one line, indicate how much additional indentation to give to the continuation line or lines.

- `Switch labels`—Indicate how much additional indentation to give case items (labels) within a switch statement.
- `Modify Space`: These choices, when enabled, indicate how many spaces you want surrounding certain items:
 - `After Keyword`—Indicate how many spaces you want to follow a keyword (reserved word or standard function). These keywords are those recognized by Hawk for Language ChromaCoding.
 - `Around Expression`—Indicate how many spaces you want surrounding an expression.
 - `After Function`—Indicate how many spaces you wish to have following the end of a function name, preceding the opening parenthesis.
 - `Around Parameter List`—Indicate how many spaces you want surrounding a function call's parameter list.
- `Enable block alignment`: Tell Hawk whether or not you want it to re-align code blocks during reformatting. If enabled, you can select which type of alignment to use. If disabled, Hawk leaves block indentation alone..



Refer to the online help topic `Align Tab` for a schematic of the alignment options and popup help descriptions for each.

- `Cuddle Else`—Tell Hawk whether or not to put the "else" portion of an "if...else" construct on a line by itself or to "cuddle" it up next to the closing brace it follows.
- `Align Function`—Often, the braces that begin and end a function definition do not conform to the same alignment rules as the braces within a function. When that is the case, leave this checkbox unchecked, and Hawk will not alter their location. To cause them to conform to the alignment type selected in this group (Unindented block, Line saver or Indented Block), just check this box.

Using the Format Feature

Once you have set the criteria for the type of file you are using (C, C++ or Java), you are ready to reformat. Complete the following steps:

1. Open the file to be formatted.
2. Select the block of code to format. If you wish to format the entire file, choose `Select All` from the `Edit` menu.
3. Once the desired section is highlighted, select the `Format Source` option from the `Text` menu. The formatting should now occur automatically, based on the settings in the `Customize -> Language -> Format` dialog.

12

Customizing the Interface



This chapter describes customizing the Hawk interface and includes the following sections:

[Duckbill Toolbars and Windows](#)

[Customizing Toolbars and Buttons](#)

[Customizing Menus](#)

[Mouse Commands](#)

[Reassigning Keys and Mouse Actions](#)

Duckbill Toolbars and Windows

Hawk has several standard toolbars, and allows more to be defined. All toolbars can be modified through a simple drag and drop interface and used in a variety of ways. For example, toolbars can be docked or free-floating, always visible or auto-hidden. The `Tools -> Customize -> Toolbars` dialog is where toolbars can be turned on, created or modified, or otherwise manipulated.

Toolbars

Several default toolbars are available in Hawk's `Customize -> Toolbars` dialog. Marking the **Visible** option for any highlighted toolbar on the `Toolbars` tab turns the toolbar on. Other toolbar options on the `Toolbars` tab include the following:

- `Hide when application is inactive` means that the toolbar is only visible when Hawk is the active application. This is usually desirable for free-floating toolbars.
- `Always on top` specifies that, when free floating, the selected toolbar or window will not allow itself to be covered up by other applications or windows.
- `Allow Docking`, when unmarked, means that the selected toolbar or window is prevented from docking. If you have decided to use a toolbar or window exclusively in free-floating mode, this will prevent accidental docking.

A brief description of the toolbars in the `Toolbars` dialog is provided here.

- `Standard Toolbar`: Turned on in Hawk by default and is docked at the top of the Hawk screen, under the main menu. A description of the `Standard Toolbar` is provided in the [The Standard Toolbar](#) in .
- `Build toolbar`: Has buttons that perform application building operations such as `compile`, `make`, `debug`, and `execute`
- `Edit Toolbar`: Has buttons that perform common editing tasks. When it is turned on, it is positioned vertically with two buttons from left to right and eight buttons down.
- `Tools Toolbar`: Has several buttons that perform various tasks, such as `File Find`, `Differences`, or `Merging`.
- `VCS Toolbar`: Has buttons for performing various operations with your defined version control program.
- `AppBasic`: This toolbar has the same items as those in the `AppBasic Window` when the `AppBasic` extension language module is loaded in the `Tools|Customize|Libraries` dialog. Many of the items on the toolbar will not be available if the `AppBasic` module is not loaded.
- `Project Toolbar/Window`: This is a window that is turned on and docked, by default, on the left-hand side of the Hawk screen. It has six tabs, `Project`, `File View`, `Outline`, `Bookmarks`, `File Open`, and `Codefolio`, which are described in various sections of this manual.
- `Output Toolbar/Window`: This is a window that is turned on by default. It initially has seven tabs and is docked on the bottom edge of the Hawk screen.

Three more tabs can optionally be added by loading Add-On for the Passaic and Per macro languages, and for the Clipboard/Scrap Viewer. Add-On are loaded using the Customize -> Libraries dialog. The tabs of the Output window are described in various chapters in this manual.

- **Debug:** This invokes the debugger command line that you defined for the project.
- **HTML Toolbar:** Has buttons that perform operations specifically designed for HTML files.
- **Telnet Toolbar/Window:** Displays a telnet window for use with the target.
- **Serial Toolbar/Window:** Displays a serial window to use as a serial console for the target.

Auto-hide Toolbars

Hawk toolbars can be optionally set to auto-hide when they are docked. When auto-hide is turned on, toolbars appear only when the mouse hovers over the area at which they are hidden. Hawk's toolbar-auto-hide option is turned on in Customize -> Toolbars -> General. To set auto-hide, mark the option in the dialog that corresponds to the edge (top, bottom, left, right) of the Hawk screen that contains the toolbar to be hidden. Any toolbars that are docked on an auto-hide-enabled edge will auto-hide.

When a toolbar is hidden a border with a small black arrow appears where the toolbar should be. Running the mouse over any part of the border causes the mouse cursor to change to an image of a hand. If the mouse cursor is allowed to maintain that shape for a brief moment, the toolbar will appear. The size of the hidden toolbar's border can be increased or decreased by changing the number for Auto-hide window size in Customize -> Toolbars -> General.

Duckbill

A duckbill window or toolbar can either be attached to one of the edges of Hawk's client area, or it can be "free-floating". When the window or toolbar is docked, it reduces the client area and does not overlay other objects. When free-floating, the window or toolbar can be placed anywhere on the screen and resized to almost any shape, but can partially obscure other objects.

Enabling and Disabling Toolbars

Once any duckbill toolbars or windows are enabled (visible), the following shortcut is available for setting the visibility status of such toolbars or windows.

1. Click with the right mouse button on any non-button part of the toolbar or window's frame.

A list of currently defined toolbars and duckbill windows will pop up. The items preceded by check marks are the ones that are currently visible.

2. To change the status of any toolbar or duckbill window, select it from the list.

As mentioned before, toolbars can also be turned on in the **Toolbars Customization** dialog. If no toolbars or dockable windows are currently visible, complete the following steps.

1. Select the **Customize** menu.
2. Choose **Toolbars**.
3. Select the desired toolbar from the list.
4. Check the **Visible** check box, and then press **Close**. The toolbar will become visible.

Docking and Moving Toolbars and Windows

You can dock or undock a toolbar or dockable window by double clicking on any non-button portion of its frame.

- If it was docked, it becomes free-floating.
- If it was free-floating, it becomes docked.

The exception is when the toolbar or window has never been docked before, and has no default docking location. In this case it won't know where to dock, and you will need to dock it manually.

Docking a Toolbar or Window Manually

You can manually dock and undock these objects by dragging them with the mouse:

- To dock a free-floating window, drag it by a part of its frame or part of its interior that is not used by any toolbar items, like buttons or the title bar. It will automatically dock when placed over the edge of the client area.
- To reposition free-floating windows without docking them, drag them by the toolbar.

When you drag a free-floating toolbar or window to the edge of Hawk's client area, you will notice a change in the mouse cursor. A small square with a plus sign in it will appear at the base of the arrow. This is to tell you that if you drop the object, it will dock. The outline of the object you are dragging will indicate what orientation it will take when dropped.

Undocking a Toolbar or Window

To undock a toolbar or window, complete the following steps.

1. Point the mouse at any non-button portion of the frame.
2. Press the left mouse button.
3. Drag the mouse. You will see an outline of the toolbar or window appear.
4. Continue to drag the object away from the edge of the Hawk client area, and then drop it wherever you want it.

- OR -

1. Click on the toolbar frame with the left mouse button.

2. Press the `Ctrl` key.
3. Release the mouse button.
4. Release the `Ctrl` key.

You can then resize and reposition the free-floating toolbar or window at will. If you have multiple toolbars docked along a single edge of the client area, you can use this drag and drop technique to change their order.

Customizing Toolbars and Buttons

Hawk has a number of standard toolbars that can be modified through a simple drag and drop interface. You can also perform the following tasks:

- Create your own toolbar with your selection of buttons.
- Modify the functions bound to each button (that will be executed when the button is pressed).
- Customize the buttons with your own bitmaps.

Adding New Toolbars

To add a new toolbar, complete the following steps:

1. Select `Customize` -> `Toolbars`. The `Toolbar Customization` dialog displays.
2. Select the `Toolbars` tab.
3. Click the `New` button. The `New Toolbar` dialog displays.
4. Enter the name of your new toolbar, and then click `OK`.

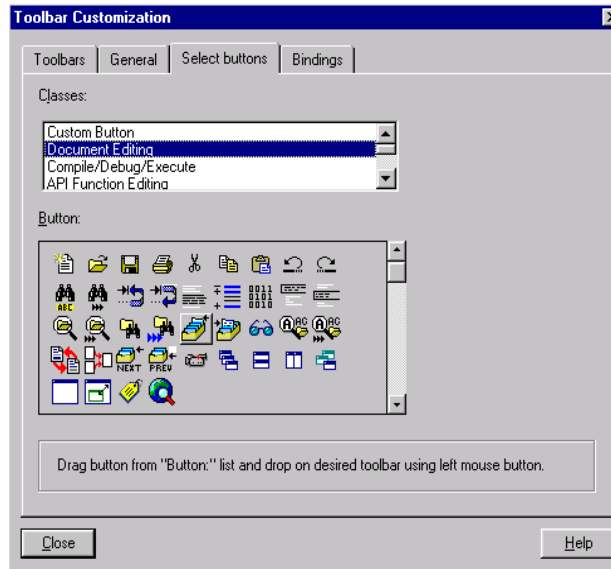
Hawk then creates a new toolbar for you. At this point the toolbar is empty. You will need to move it and dock it. You will also need to add buttons to it. The latter is covered in the next topic, [Adding and Changing Toolbar Buttons](#). The toolbar information is stored in the `MWHAWK.INI` file.

Adding and Changing Toolbar Buttons

To add or change a toolbar button, complete the following steps.

1. Go to the `Customize` -> `Toolbars` dialog.
2. Click the `Toolbars` tab.
3. Select the toolbar you wish to modify; the toolbar will be displayed in a separate window.
4. Click the `Select buttons` tab. On the `Select buttons` tab of the `Toolbar Customization` dialog, Hawk enters a special mode where the toolbars are temporarily inoperative.

Figure 12-1. Toolbar Customization: Select Buttons Screen



5. From this dialog you can perform the following tasks:
 - Drag and drop buttons from the dialog to any toolbar.
 - Drag buttons from one toolbar to another.
 - Rearrange buttons within a toolbar, dragging them into the desired position.
 - Remove a button simply by dragging it off the toolbar.

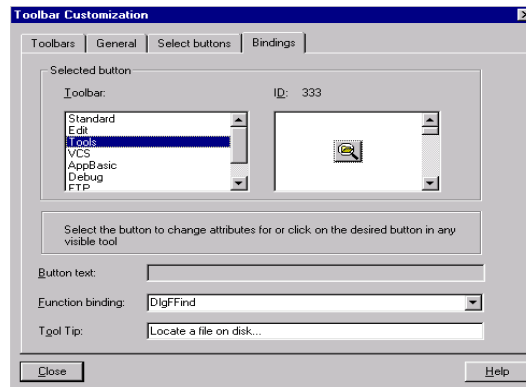
There are several categories of buttons in the dialog, listed in the `Classes` window. The category `User-Defined Tools` includes buttons with no pre-defined function bindings, to which you can quickly bind your function.

Binding a Function to a Button

To bind a function to a toolbar button:

1. Go to the `Customize` -> `Toolbars` dialog.
2. Click on the `Bindings` tab.

Figure 12-2. Toolbar Customization: Bindings Screen



3. Select a toolbar from the list box.
4. Cycle through the buttons on that toolbar until you get to the one you want to change.
5. Make necessary entries in the following edit boxes for each button:
 - The `Button text` field only applies to buttons that contain text, not bitmaps.
 - The `Function binding` field contains the function call to be executed when the button is pressed. This function must be available to Hawk and follow the rules of `LibFunctionExec`.
 - The `Tool Tip` field contains the message that pops up when the mouse cursor pauses over the button.
 - The button information, such as the toolbar it belongs to, its position on the toolbar, the source of the bitmap image and the function bindings, is stored in a file called `CWRIGHT.BTN` in the main Hawk directory. Hawk takes care of updating this file, much like an `.INI` file, as you customize your buttons..



All bitmaps that are available in the dialog for the buttons come from the `CWDDL32.DLL`. If you have your own bitmaps you would like to add, you will need to load your DLL containing the images.

Combo Box History Lists

Many of Hawk's prompts have combo box history lists that store previous responses made at the prompt. Most notably, the `Command Key` prompt, the `Search` and `Replace` prompts, and the prompt at which you enter the name of a file to edit, have prompt histories. The lists can be accessed by either bringing down the drop-down list under the combo, or by pressing the up arrow or the down arrow. Pressing the up arrow reuses the most recently entered response, whereas pressing the down arrow reuses the oldest. Repeatedly pressing the up arrow allows you to view earlier responses. Repeatedly pressing the down arrow key views more recent responses. Pressing `Enter` accepts the response displayed.

As each historical response is displayed, you will note that it is highlighted. If you issue an editing command, such as `[Home]`, `[Ins]` or `[End]` the highlighting disappears and

you are permitted to continue editing the response. You can accept the edited response at anytime by pressing **Enter**. If, when the response is still highlighted, you type characters, the typing replaces the highlighted text. This allows you to easily type in a new response, if the one you are looking for is not in the history.

Editing Combo Box History Lists

There is a history editor on the **History** tab of the **Environment** dialog on the **Customize** menu. It allows you to view and remove members of the prompt histories. It also lets you adjust the size of the lists.

Many prompt histories are saved between Hawk sessions by default. The information is saved in the `state` file..



[Chapter 17 Configuration Files and Command Line Parameters](#) contains additional information about Hawk `state` files.

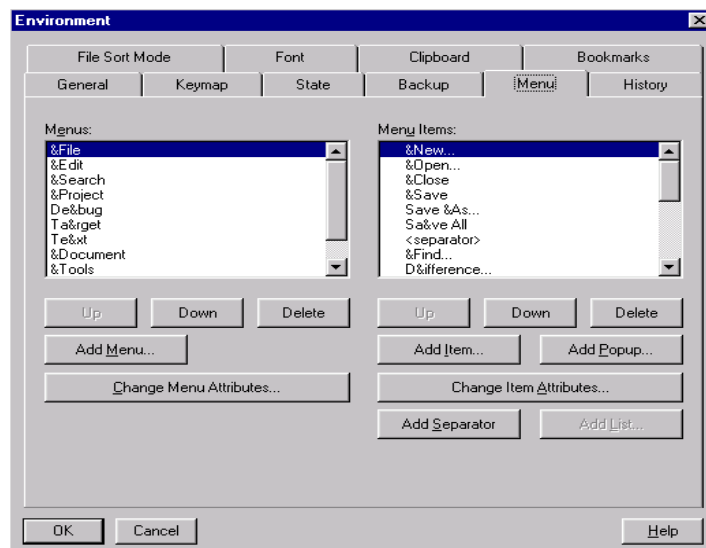
Customizing Menus

This section describes how to edit standard menus, submenus, menu items and pop-up menus.

Menu Editor

You can change, delete and add menus and menu items through the **Menu Editor** on the **Customize** -> **Environment** -> **Menu** dialog.

Figure 12-3. Environment Dialog: Menu Tab Screen



The dialog list **Menus** and **Menu Items**, as described in the following topics.

Menus

Menus are the top-most selections on the `Menu` bar. They contain items and submenus. You begin your modification of the menu by selecting a menu from the list on the left.

- `Up/Down`: Allow you to change the position of the menu relative to other menus. `Up` and `down` refer to the position in the list box. `Up` moves the selected menu to the left on the menu bar, while `Down` moves the menu to the right.
- `Add Menu`: Lets you add a menu to Hawk's menu bar. You must provide a name for the menu, a helpful message to display when the menu is selected, and, optionally, one or more functions to initialize the menu. The initialization functions are responsible for disabling or check marking menu items as necessary.
- `Change Menu Attributes`: Similar to the `Add Menu` dialog, except that the attributes for the selected menu are displayed for editing. You cannot change the ID of a menu.
- `Delete`: Deletes the selected menu. You could delete all menus, if you wish. You would then find that you are unable to add them back interactively, once you leave this dialog. (You couldn't get the dialog back.) If the worst happens, you can restore the original menu by editing Hawk's configuration file, `MWHAWK.INI`. Locate the `[Menu]` section of the file and rename the section, or delete all lines under that section heading.

Menu Items and Submenus

Menu items and submenus are the entries listed on a menu. After selecting a menu, the items on that menu will appear in the list box on the right. Any submenus on the menu will also be listed in order, preceded by a `+` sign. You may then select a menu item or submenu from the list on which to operate.

- `Up/Down`: Allow you to change the position of the item or submenu on the menu. You may also move separator lines with these buttons.
- `Add Item`: Lets you add an item to one of Hawk's menus. You must provide a name for the item (Menu item text), a helpful message to display when the menu is selected (Help string), and a function to execute when that item is selected (Handler function). If there is a short-cut keystroke that performs the same function, you may enter that keystroke for display on the menu (Key string).
- `Add Popup`: Brings up a dialog similar to the `Add Menu` dialog. Enter a name for the submenu a help message, and any functions to initialize the submenu.
- `Change Item Attributes`: The operation of this button depends on whether or not you have selected a submenu or a normal menu item. If you have selected a normal menu item, the `Change Item Attributes` dialog is like the `Add Item` dialog, except that the attributes for the selected item are displayed for editing. If you have selected a submenu, the dialog is like the `Add Popup` dialog.
- `Delete`: Deletes the selected item. You could delete all items in a menu, if you wish. You can also delete separators with this button.

- **Add Separator:** Allows you to add a separator line at the selected position within the menu.
- **Add List:** There are five lists which may be added to the bottom of a menu: recent files, scrap buffers, edit buffers, recent projects, and windows. If you are using `One-document-per-window` mode, the list of edit buffers is not available.

Each of these lists may only appear on one menu. Therefore, you must first delete a list from a menu before you can add it to another. This button is disabled if there are no unassigned lists.

Operating on Submenus

To view or modify a submenu, you must double click with the mouse on the submenu (marked with a plus) in the list on the right. The submenu then moves to the list on the left and is marked with a minus sign. You may then operate on it as you would a menu, adding items, moving and deleting items as you desire.

Whenever you are through operating on a submenu, you may move it back to the list on the right by double clicking on the name again. The minus sign will change back to a plus when the submenu arrives in the list on the right.

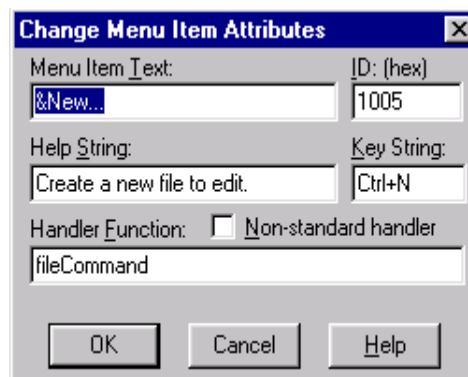
This system allows you to have any level of nesting of submenus that you like. Just keep adding them on the right and moving them over to the left.

Changing the Functionality of a Menu Item

To change the function associated with a particular menu item, complete the following steps:

1. Select the `Customize -> Environment` dialog.
2. Click the `Menu` tab.
3. Click a menu, and corresponding menu item to change.
4. Click the `Change Item Attributes` button.

Figure 12-4. Change Menu Item Attributes Dialog



This dialog then shows you information about that menu item such as its ID in Hex, the function bound to it, and so forth. Before you start customizing, you will find that the existing menu items are bound to functions that begin with a lowercase letter. These functions are not accessible directly by you. They require special menu handlers and can only be called from a DLL.

To successfully change the functionality of a menu item, you must:

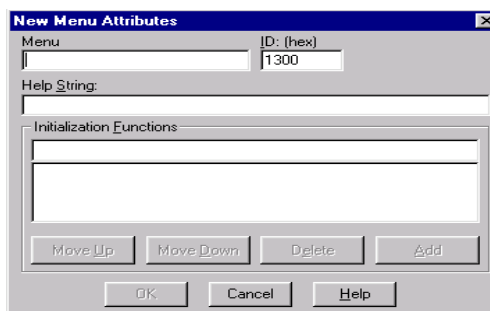
- Bind the menu item to a function that begins with an uppercase letter. Any of the Hawk API functions should work, or your own functions (or macros), provided they have met the criteria to be recognized by Hawk.
- Check the small check box labeled **Non-standard handler** when binding functions that do not have a menu handler. You must check this box, or Hawk will not handle your customization, and your menu items will not work.

Adding a Menu Item

To add a new menu item, complete the following steps:

1. Select the **Customize -> Environment** dialog.
2. Click the **Menu** tab.
3. Select the menu to which the item should be added.
4. Click the **Add Item** button. The **New Menu Item Attributes** dialog displays. This dialog is the same as the **Change Item Attributes** dialog.

Figure 12-5. New Menu Item Attributes Dialog



5. In the **Menu Item Text** field, enter the desired text for the menu item.
6. In the **Help String** field, enter text for the menu item's tool tip.
7. In the **Key String** field, enter a keyboard shortcut for this menu item.



Entering a key string in the **Add Menu Item Attributes** dialog does not make the key string work. It must first be defined in the **Customize -> Keyboard** dialog. To learn more about customizing keystrokes, see the topic [Reassigning Keys and Mouse Actions](#).

8. In the **Handler Function** field, enter the function binding, beginning with an uppercase letter.
9. Select the check box labeled **Non-standard handler**.

10. Click **OK**. Hawk assigns the menu ID in hex to the new item.

Example: To add Hawk's Perl help file to your Help menu, complete the following:

1. Select the **Customize** -> **Environment** dialog.
2. Click the **Menu** tab.
3. Select the **&Help** menu under Menus.
4. Click the **Add Item** button.
5. In the Menu Item Text field, enter: **Perl Help File**.
6. In the Help String field, enter **Perl Help**.
7. In the Key String field, enter **Ctrl-Q**.
8. In the Handler Function field, enter the following, then click **OK**:

```
ExecUserCmd='winhelp c:\mwos\dos\bin\Perl.hlp',1.
```



If you prefer to use keystrokes instead of menus, any menu item can be bound to a keystroke. Refer to the topic [Reassigning Keys and Mouse Actions](#), in this chapter.

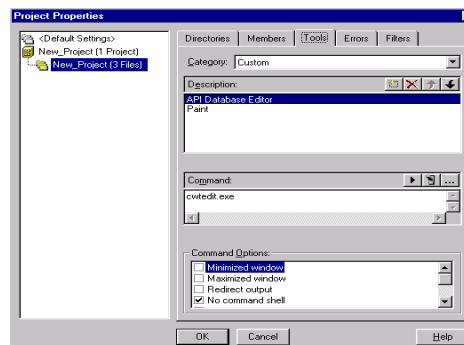
Customizing External Operations within Hawk

As described in [Chapter 7 Projects, Project Spaces, and Workspaces](#), Hawk provides the ability to use external operations from within the editor. Any program can be added as menu items to the bottom of the **Tools** menu. The added items will be numbered. This is a quick and handy way to add links to your favorite applications, or execute custom compiles or other external operations.

To add a custom item to the **Tools** Menu:

1. Select the **Project** -> **Settings** dialog.
2. Click the **Tools** tab.
3. Choose the **Custom** option from the drop down under the **Category** combo box. The dialog is displayed next.

Figure 12-6. Project Properties Dialog – Custom Category



4. In the **Command:** box, enter the command or executable file that the menu item should launch. Press the right arrow for a list of pre-defined macros; press the **Browse** button to search for executable files.

Remember that Hawk will shell to DOS in the directory specified on `Project -> Settings -> Directories -> Working Directory` when executing your command line, so you may need to provide the complete path.

User-Definable Popup Menus

Positioning the mouse in certain areas within Hawk and right clicking will bring up a popup menu. You can change the way the popup menus work or add your own sections to them using Hawk's `Popup Menu Editor`. The following paragraphs describe Hawk's `Popup Menu Editor` and the steps for creating and editing popup menus.

Hawk's `Popup Menu Editor` is for editing and creating popup menus. The changes are applied to designated files containing menu descriptions and are effective immediately. Hawk's default menu file is `MWHAWK.MNU`, but other menu description files can also be created and modified with the editor.



For more information about the syntax used for creating and editing popup menus and menu items, see the topic *Popup Menu Semantics*.

Clicking the `Edit this menu` item on most Hawk popup menus accesses the `Popup Menu Editor`. The function `DlgEditPopupMenu`, used from Hawk's `API Command` dialog, a keystroke, button, or menu item, will also access the editor.

As mentioned, changes made in the `Popup Menu Editor` are applied to whatever file contains the settings for the selected popup menu. In most cases this file is the `MWHAWK.MNU` file, Hawk's main popup menu configuration file. `MWHAWK.MNU` contains sections similar to a `Windows .INI` file. Each section begins with a section heading, a word or label enclosed in square brackets. The lines that follow the section heading (`Item Definition Lines`) describe the menu items that will appear on the popup menu, and what happens when you select each item from the menu.

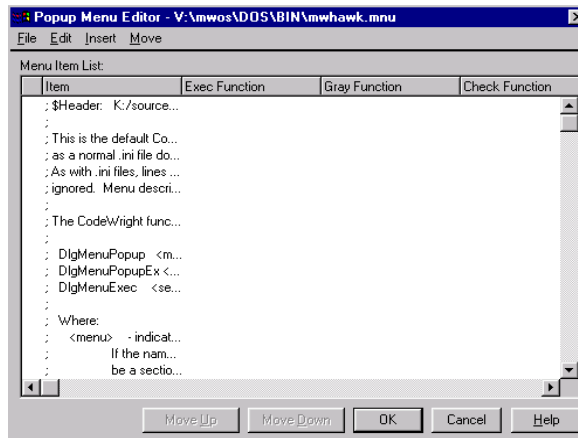
Editing or Creating a Popup Menu

Complete the following steps to edit or create a popup menu in Hawk:

1. In `Tools -> API Command`, type `DlgEditPopupMenu`.

The `Popup Menu Editor` should appear with Hawk's default menu definition file `MWHAWK.MNU` open.

Figure 12-7. Popup Menu Editor

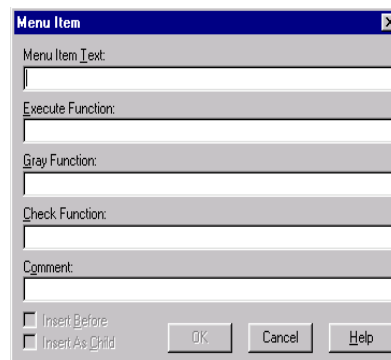


2. In the **Popup Menu Editor**, click **File** -> **New Menu** to create a new menu, or click **File** -> **Open Menu File** to open an existing menu. If a new menu is being created, a new, empty menu will be opened in the editor. If an existing menu is opened, the existing menu will appear in the editor.
 - When opening an existing popup menu for editing, the **Open Menu** or **Menu File** dialog appears. The name of the menu description file (for example **MWHAWK.MNU**) has to be specified in the **File** box, and the name of the menu (for example **[Utilities]**) has to be specified in the **Menu** box.

Complete the following steps to add or modify a menu item in the popup menu:

1. Click **Insert** -> **Menu Item** to add a menu item. Click **Edit** -> **Modify** to modify a selected menu item. In both cases, the **Menu Item** dialog appears.

Figure 12-8. Menu Item Dialog



2. Fill in the necessary fields in the **Menu Item** dialog. The uses for the fields are as follows:
 - **Menu Item Text** : The text that is seen on the popup menu. Any character in the text that is preceded by the ampersand (&) acts as a key accelerator for the item. Pressing the key for that character while the menu is visible accesses the item.

- **Execute Function:** For menu items that are to appear as normal menu items. The functions are called when the menu item is clicked. Most exported Hawk functions can be used.

Example: The following item can be added to any popup menu for opening files:

```
&Open File BufEditFile
```

Where `&Open File` is the menu item string, and `BufEditFile` is the function to execute.

- **Gray Function:** For menu items that are grayed out if they are disabled or if they do not apply to the state of the environment from which the popup menu was accessed.

Example: The `Terminate` item on the `Perl` popup menu is gray when a `Perl` macro is not running. The menu item is used to forcibly terminate a running `Perl` macro.

```
&Terminate_PerlCtrl 0!_PerlCtrl -1
```

Where `&Terminate` is the menu item, `!_PerlCtrl-1` is the function that is called when the menu is first brought up, and `PerlCtrl-0` is the state of the function available when the menu item is enabled. The exclamation point (!) that the complement of the function's return value should be used when the function is called.

- **Check Function:** The `Check Function` is for menu items that show check marks when they are enabled.

Example: The `Debug Mode` item on the `Perl` popup menu, `IDH Perl Popup Menu`, toggles the `Perl IDH Getting Started with Perl for Hawk interpreter's Debug mode IDH Execute in Debug Mode`. The item is not checked if `Debug Mode` is turned off.

```
&Debug ModePerlFlags 2 -2PerlFlags 2 -1
```

Where `&Debug` is the menu item, `PerlFlags 2` is the function that is called when the menu is first brought up and `PerlFlags 2 -1` is the state of the function when the menu item is checked.

- **Comment :** Comments can be inserted for each menu item added. They are not displayed on the menu. They can be used as reminders or notes for the menu item.

Popup Menu Semantics

Changes made in the `Popup Menu Editor` are applied to specified menu description files. Hawk's default menu description file is stored in the Hawk installation directory and is called `MWHAWK.MNU`. It contains sections as a normal `.INI` file does except that the contents are menu descriptors. As with `.INI` files, lines beginning with a semicolon and blank lines are ignored. Before creating or modifying popup menus in Hawk, there are some things to know about the semantics of the information contained in popup menu description files.

Refer to the following paragraphs for information about popup menu semantics. Note that all of the items that follow are valid when used from within the `Popup Menu Editor`.

Creating a Menu Item

Keep in mind the following when creating a new menu item in `Menu Item` dialog of the `Popup Menu Editor`:

- Beginning Check Functions or Gray Functions with an exclamation mark (!), complements the value returned by the function being used. This can be useful when it becomes necessary to access the opposite of the function's return value.
- If a semicolon is required in any of the function strings it must be escaped by preceding it with a backslash. The two character sequence '\;' will be replaced by a semicolon prior to execution of the function string.

Terminating Menu Sections

If you look at the `MWHAWK.MNU` file, you will notice that individual popup menus are contained in menu sections (designated by headings that are contained in square brackets ' [] '), each of which ends with a special line containing

```
[###]
```

The `[###]` allows the menu editor to delete a menu description and write a new one without also deleting comments that may precede the menu description that immediately follows the one being modified. Because this special line looks like a section name it terminates the process that is looking for the end of a section but it is otherwise benign.

Conditionals

A conditional mechanism similar to the C/C++ `#if/#else/#endif` is possible when creating new popup menu items. There is a conditional item on `Insert menu` of the `Popup Menu Editor` that allows conditional items to be created. The general form for popup menu conditionals is:

```
#if testFunction
menu description lines
#elseif testFunction
menu description lines
#else
menu description lines
#endif
```

The `<test function>` element is a function, similar to a `Check` function or a `Gray` function, whose return value is used to determine whether or not to include the following `<menu description lines>` in the popup. The conditionals may be nested to an arbitrary depth.

As a shorthand notation, if the `<test function>` begins with a left parenthesis, the entire `<test function>` string is passed to `MacroEvaluate`. This shortcut applies to all

entries requiring a function spec: `Exec` functions, `Gray` functions, and `Check` functions. Example:

```
#if ClipboardQHasText
Paste From Clipboard
#endif
```

'Include' mechanism

Sometimes different popup menus have common elements and it is useful to be able to define those elements once and use that description many times. This is possible using a special menu item in one of three forms:

```
@file
@file [menuName]
@[menuName]
```

The first form specifies a file to be included in its entirety. The second form indicates a specific menu description contained in the specified file. The third form specifies only the name of a menu; it assumes that the menu is in the file containing the menu description line.

Dynamic Menu Generation

It is possible to specify, directly in a menu description, the name of a function to call to generate a menu description string (the string that is seen on the menu). The returned string, which should be an allocated string, is then used as if it had been placed in the menu description at that point. The syntax for specifying this action is:

```
*functionName
```

The entire line beyond the introductory asterisk is taken to be a function name and its arguments. As with other menu functions, if the generation function begins with a left parenthesis the entire line is passed to `MacroEvaluate`. Note that this item may be used on menus that have other, explicitly defined, entries or it may be the only item in the menu definition.

Supporting Functions

There are two functions that support the operation of popup menus. They are as follows:

- `DlgMenuPopup` assigns a popup menu to a key or mouse click. The function is formatted as the following:

```
DlgMenuPopup <menu>
```

- `DlgMenuExec` executes other sections within the `MWHAWK.MNU` file. The function is formatted as the following:

```
DlgMenuExec <section>
```

Where:

`<menu>` — If the name starts with '`[`', it is assumed to be a section in the `.MENU` file, otherwise it is the name of the file to use (no sections in file).

`<section>` — A normal Hawk section to execute out of the `.MENU` file. Use `ConfigFileRead()` to read out of other files.

Refer to the following example:

Example: The `[Utilities]` section can be bound to shift-right-mouse click through the following key binding command (see the topic in this chapter [Customizing with Keybindings](#) for more information):

```
KmapAssign='<Shift-Mouse_right_click>' `DlgMenuPopup
[utilities]`
```

Using Keymaps

The Hawk editor is several editors in one. In its "standard" mode, the Hawk keymap uses the Common User Access (CUA) key-command set. If you have ever used a Windows editor before, such as NotePad or SysEdit, you will find that you can guess many of the basic key commands. This command set offers a number of short-cut keystrokes to bypass menus and perform various other tasks.

Hawk offers alternate keymap command sets as well. These command sets may be more to the liking of users whose roots are embedded in DOS, UNIX and other non-Windows platforms. These command sets have been supplemented with a number of CUA commands.

In all, Hawk ships with four keymaps. The command sets are usually chosen at the time of installation, but they can also be switched on the fly in the `Customize -> Environment -> Keymap` dialog. Command sets may be selected from the following choices.

- CUA command set
- BRIEF command set (emulates the BRIEF editor)
- vi command set (emulates the vi editor found on UNIX systems)
- Epsilon command set (emulates the Emacs-derived Epsilon editor from Lugaru)

You can modify the functions that are executed when a keystroke is pressed in the `Customize -> Keyboard` dialog (see [Binding Keystrokes to Functions or Macros](#) in this chapter).

Further discussion of the CUA and BRIEF command sets follow.

CUA Key Commands

The CUA keymap has many more assignments in it than are covered by the Common User Access standard. In devising these additional keystrokes, we have relied heavily on mnemonics, that is, keys that easily form an association in your memory with the action they perform. This makes the commands easier to learn and remember.

If you are familiar with CUA commands, you may doubt that CUA commands would be easy to associate, and some commands are not. You will find, however, that most commands of this variety have a more memorable equivalent.

Example: To search again you have your choice of # or `CSs`. To find, press `Cf`.

BRIEF Key Commands

BRIEF emulates the BRIEF editor that is popular in the DOS environment. BRIEF key commands are not CUA compliant. This means that they conflict with the way Windows commands normally work.

Example: The BRIEF commands make extensive use of `Alt` key combinations. CUA rules reserve most of these for standard commands, such as accessing menus. For instance, `Alt-F` is usually reserved for bringing down the `File` menu. In BRIEF, this is used for displaying the output file's name on the status line.

For this reason, when using the BRIEF command set, you must press and release the `H` key to access the menu. To access the `File` menu, for example, you press and release the `H` key and then press `f`. Similarly, to access other menus, you press and release `H` and then press the underlined letter in the menu's name.

Mouse Commands

This chapter describes using the various mouse commands in Hawk.

Mouse Scrolling Speed

If you begin selections with the mouse and then move the mouse outside the visible window, the window will begin scrolling. Depending on how far out of the window the mouse is, Hawk will vary that scroll rate, specifically, the farther out the mouse is, the faster the scrolling.

In some cases, the space available to move the mouse beyond the window edge is limited, such as full screen mode. Hawk provides some methods for gaining control over scrolling speed in these cases. Control is gained with the `Ctrl`, `Shift`, and `Alt` keys under the following conditions:

- If none of the keys are pressed, scrolling speed is the default.
- If one of the keys is pressed, scrolling speed is twice as fast.
- If two keys are pressed, scrolling speed is four times as fast.
- If three keys are pressed, scrolling speed is eight times as fast.

Inclusive or Exclusive Selection

In the standard keymaps `CUA`, `BRIEF` and `vi`, you can select text by clicking with the left mouse button and dragging the mouse across the area you want to select. When the character at the cursor is included in the block, the block is considered inclusive. Default settings are as follows:

- In the `vi` keymap, inclusive block selection is the default.

- In the `CUA` and `BRIEF` keymaps, exclusive block selection is the default (the character at the cursor is not included in the block).

Closed Selections

You can elect to have selections that you make with the mouse be either closed or open when you release the mouse button. A closed selection means you will not change the selection size or shape when you move the cursor. Your keymap dictates the initial setting for this option; you can change the initial setting with the `Leave Mouse Selections Open` option on the `Customize -> Environment -> Keymap` dialog.

Selections made with key commands are usually open; one end of the selection is defined by the cursor position, and the selection moves with the cursor. Your keymap may have a command to toggle open or closed a mouse selection or other selection.

Examples:

In the `BRIEF`-compatible keymap, toggle the selection open or closed using the `Fa` command.

In the `CUA` keymap, toggle the selection open or closed using the `F.` command.

To expand a closed selection, move the cursor to the new endpoint desired, and use the applicable key command to toggle the selection open.

- The distinction between closed and open selections is not meaningful in the `CUA` keymap unless you have turned on persistent selections. Otherwise, `CUA` removes the selection, whether closed or open, whenever you execute a cursor motion command.

Column Marking

To make a column selection with the mouse, just click and drag with the right mouse button instead of the left. Column blocks are always inclusive, regardless of which keymap you are using.

Line Selections

There is an adjustable margin between the left edge of the buffer and the window border. You can use this area for making line selections:

- Click with the mouse in this space to select the line to the right.
- Click and drag the mouse to select a series of lines — even if the mouse happens to stray from the margin.




Word Selections

As with many Windows editors, double clicking on a word makes a selection encompassing that word. If you continue to hold down the button on the second click and drag it around, you can select in units of words. You may be familiar with this method of selection from one of several word processors.

Status Line Actions

There are a number of functions or pop-up menus that can be accessed just by clicking with the right mouse button on hotspots on the status line. These actions are listed below:

Table 12-1. Status Line Actions

Action	Description
Insert/Overtyping Toggle	Toggle between insert and overtype mode by right-clicking on the <code>Ins</code> or <code>Ovr</code> designation on the status line.
Read-only /Read-write Toggle	Toggle between Read-only and Read-write status by right-clicking in the box to the left of the line number. The box may be blank or have a "RO" designation in it. This toggles both file and buffer status.
Go to line	Bring up the <code>Go to Line</code> dialog by right-clicking in the <code>Line number</code> box.
Next Message	Have Hawk process the next <code>Build error</code> message by right-clicking in the message box at the left of the status line.
	Bring up a customization pop-up menu with options for accessing various dialogs used for customizing Hawk.
	Bring up a <code>Word Wrap Options</code> menu to quickly toggle wrap mode and customize wrap options.
	Bring up a <code>CodeSense</code> customization menu to quickly turn on and customize Hawk's <code>CodeSense</code> and <code>Outline Symbols</code> features. For more information about these features see Chapter 6 Editing Features and Chapter 10 Search and Replace and Other Navigational Tools .

A related command is the right mouse click on the `Toolbar Search` combo box, located on the `Standard Toolbar` instead of the status line. Right-click here to bring up the `Search Options` dialog, allowing you to check or change the settings in preparation for using the `Toolbar Search` or other search mechanism.

Text Drag and Drop

After you have selected a block, you can move or copy the block by dragging the block to its new location. Text can even be moved between open windows.

Use the following drag and drop operations:

- To move the block complete the following:
 1. Place the mouse cursor in the selection's highlighted area.
 2. Press the left mouse button. The cursor will change shape to indicate that a drag and drop operation has commenced.
 3. Continue to press the mouse button, moving the mouse cursor to the intended destination.
 4. The caret (text cursor) follows the mouse cursor, indicating where the destination would be at any given moment. When the caret is at the intended destination, release the mouse button and the selected text moves there.
- To move text between windows, complete the steps above but press **Shift** as you drag. Be sure to click and hold the mouse button before pressing the **Shift** key. Then drag and release the text.
- To perform a copy operation, complete the steps above, but press the **Ctrl-Insert** key when you initiate the operation.
- To cancel a drag and drop operation after it has been commenced, move the mouse cursor back over the selection and release the mouse button. A selection cannot be copied or moved onto itself. The operation is then cancelled and the selection removed.
- To disable text drag and drop, clear the **Drag and drop text with mouse** check box on the **Customize -> Environment -> General** dialog.

Mouse Copy and Move

In addition to **Text Drag and Drop**, Hawk supports another method of copying and moving text with the mouse:

1. Select the text you want to copy or move.
2. Point the mouse at the place you want it to go.
3. Issue the proper mouse command. The command for moving the selected text is **F6**. The copy command is **FS6**.

Sometimes, when you are moving or copying text, the source and destination of the text are too far apart to be viewed in the same window at the same time. You can still use the mouse move and copy commands, by opening up another window, or scrolling between source and destination:

- You might have several sections of text that you want to copy or move, all from the same source location to a single destination. In this case, viewing the source in one window and the destination in another is the best approach.
- If you have just one copy or move operation in mind, you can select the desired text and then scroll to the destination. The selection will scroll off the screen, but as long as you are using closed selections for mouse operations, it will not change. Refer to the topic [Closed Selections](#) in this chapter.

Creating Windows with a Mouse

It is very easy to create windows of an arbitrary size and position when you are working with a mouse. This is the quickest way to open up a second window on the current buffer, so that you can view two different sections of the buffer at the same time.

To create a window, complete the following steps:

1. Point to any part of Hawk's `Client Area` that is not currently occupied by a window.
2. Click with the left button and drag the mouse cursor to any other part of the `Client Area`.

Whatever buffer was current prior to creating the window is also made visible in the new window. This means you will have at least two windows in which to view the current buffer.

If you do not like this use of the left mouse button, you can turn it off on the `Customize -> Environment -> General` dialog. Clear the `Create Windows Using Mouse` check box.

Drag-and-Drop File Loading

The `Drag and Drop` feature allows you to load files into applications from the `Windows File Manager`. Select one or more files listed in the `File Manager` and drag them with the mouse to a minimized application. When the mouse button is released, the application is restored, and the file or files are loaded or processed.

If you already have files loaded in Hawk, the files you drag and drop onto Hawk will be added to those already being edited:

- A window is created for each of the files dropped onto Hawk if you have checked the box `One document per window` on the `Customize -> Environment -> General` dialog.
- If windows are not created, the window that was current will contain the first file of those dropped.

Expand / Collapse Selective Display

When you are using selective display mode, you can use a mouse for expanding and collapsing sections of text as you might the sections of an outline.



Refer to the topic [Selective Display](#) in [Chapter 10](#) for more information about this use of the mouse.

Reassigning Keys and Mouse Actions

Refer to the following discussion of macros and keybindings.

Keymap-Specific Assignments

This section describes making keymap assignments specific to your keymap. Your main tool for doing this will be Hawk's `Assign Keys` dialog. Keys can be modified using Hawk API functions, keystroke macros, `AppBasic`, `Perl`, or `API Macros`, or your own functions from custom DLLs. Remember that custom functions need to be exported with a `LibExport` call in the DLL's `_init` function in order for the key assignment to work. Otherwise, a "function not found" message will appear in Hawk's status bar when the keystroke is pressed.



Refer to [Chapter 15 Extending Hawk](#) for more information on custom DLLs or Add-Ons.

Customizing with Keybindings

As mentioned in the section on [Using Keymaps](#), Hawk ships with four unique keymaps. These are `CUA`, the Windows interface; `BRIEF`, for those used to the old DOS `BRIEF` editor; `EPSILON`, and `vi` for those who come from a UNIX background. The functions that will be executed when you press a key will depend upon which of these keymaps you are in. You can also create your own keymaps.

If you are not happy with the way a certain operation is performed within a keymap, you may want to try that operation in another keymap (Select `Customize` -> `Environment` -> `Keymap` and change the keymap selection for testing purposes). If you find that another keymap has a function that works more appropriately, you can replace the original keymap's function with the preferred function. The change is made in the `Customize` -> `Keyboard` dialog. See [Binding Keystrokes to Functions or Macros](#), later in this chapter.

Keystroke Recording/Playback

Keystroke macros consist of a series of keystrokes that can be run sequentially as a unit. The following sections describe the process for making keystroke macros in Hawk.

Recording a Keystroke Macro

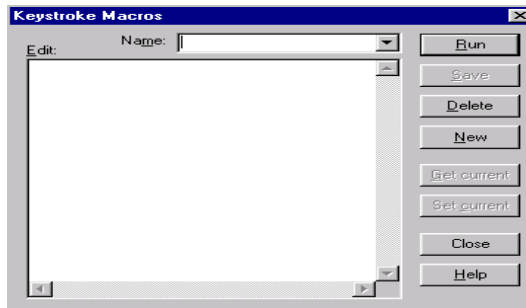
You can record a keystroke macro in any of the following ways:

- Press `&` (or `Cx` in `BRIEF`) to begin and end a recording.
- Select `Edit` -> `Record` to begin and end a recording.
- Type the ASCII representations for a series of keystrokes directly into the `Keystroke Macros` dialog (accessed by choosing `Edit` -> `Keystroke Macros`).

Saving a Macro

The `Keystroke Macros` dialog is used to create, edit, save, and delete named keystroke macros.

Figure 12-9. Keystroke Macros Editor



To make a named keystroke macro, complete the following steps:

1. Record a series of keystrokes using the methods described in the [Recording a Keystroke Macro](#) topic above.
2. Load the recorded macro into the Keystroke Macros Editor by clicking **New** and then **Get Current**. The keystroke macro that was just-recorded will be displayed in the editor.
3. Type a name into the Name combo-box.
4. Click **Save**.

Once the keystroke macro has been saved with a name, it can be used from Hawk's Tools -> API Command dialog, or bound to a keystroke, button, or menu-item simply by typing the name of the macro in the appropriate place. Named keystroke macros can also be used in Hawk's AppBasic macros, Perl macros, and API macros.



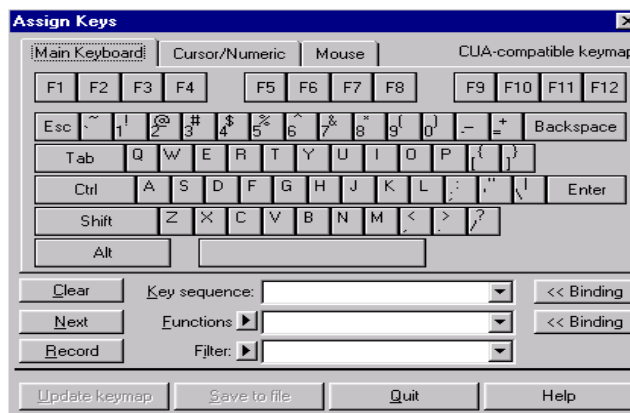
For more information on macros, see [Chapter 15 Extending Hawk](#).


Binding Keystrokes to Functions or Macros

To bind a keystroke to a function or macro, complete the following steps:

1. Go to the **Customize** -> **Keyboard** dialog.

Figure 12-10. Tools -> Customize -> Keyboard Dialog



2. Enter a `Key Sequence` value to bind to the named function or macro.
 - The easiest way to do this is to select the `Main Keyboard` tab, and click the key representations in the proper sequence.
 - Verify that the key sequence isn't already assigned by pressing the `Binding` button next to the `Key Sequence` box.
3. Locate the macro (or function) to bind from the drop-down list in the `Functions` box. To narrow your search, you can do one of the following:
 - Choose or enter a filter (keymap-related) in the `Filter` box. For example, to see only functions that are available from the CUA keymap DLL, type `CUA_`. Then when you use the drop-down list under the `Functions` combo box you will only see functions that begin with `CUA_`.
 - Press the right arrow  to the left of the `Functions` box, and choose either `Functions` or one of the macro categories. This will cause only the functions in the selected category to display in the drop-down list under the `Functions` combo box.
4. Verify that the function or macro does not already have a key binding by pressing the `Binding` button to the right of the `Functions` box.
5. Once the appropriate key sequence and function/macro are displayed:
 - Press `Update Keymap` to save the keybinding for the current session only,
 - OR -
 - Press `Save to file` to save the keybinding for continued use.

13

File Loading and Backup



This chapter describes options and strategies for file loading and validation, and for file backup and auto-save. The following sections are included:

[File Loading, Reloading and Validation](#)

[File Backups and Auto-save](#)

[Making Files Read-only](#)

File Loading, Reloading and Validation

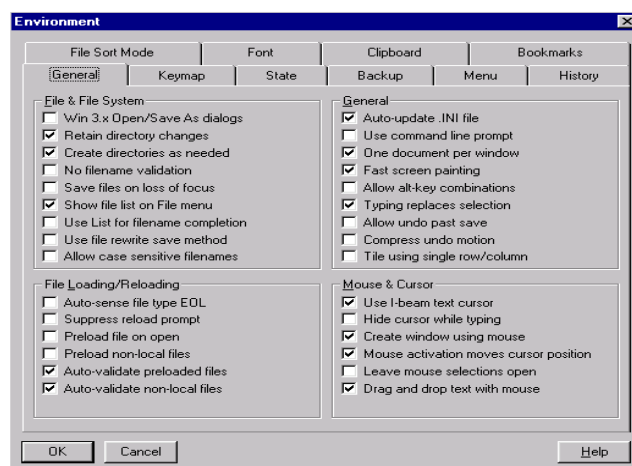
Hawk has options for preloading and auto-validating local and non-local files. This section describes these terms, and how to use the features that enable them.

- A file is preloaded in Hawk if it is fully loaded into virtual memory, without relying on pointers to the actual file on disk. This method for loading files is optional and must be turned on manually. It takes longer, and uses more temporary disk storage, but does not rely on the presence of the original file. If a network connection is lost or if the file is somehow deleted, you will still be able to continue.
- A file is validated in Hawk when periodic checks are made to see if the file being edited has been altered by another process. Auto-validation can be a process on some remote media, in which case, it may be more efficient to turn it off.

To set file loading and validation options:

1. Select the **Customize** -> **Environment** dialog.
2. Click the **General** tab.

Figure 13-1. Customize -> Environment -> General Tab



3. The following options in the **File Loading/Reloading** group can be enabled as precautionary measures to maintain the integrity of files loaded in Hawk:

- **Auto-detect Unicode Files:** Mark **Auto-detect Unicode files** in **Customize -> Environment -> General** to turn on auto-detection and bidirectional conversion of code that follows the Unicode standard. Note that this checkbox only affects files that are loaded after its state has been changed. To open a file in the opposite state, the file must first be closed, the state changed, and then the file must be opened again.

When this option is marked, the first portion of a file is examined when the file is opened. If the first word is the Unicode signature (Oxfeff), and the remaining words all convert to single-byte characters, the file is determined to be Unicode. As the remainder of the file is read in, each Unicode character is converted to a single-byte character. When the file is written out, the Unicode signature is written first and the single bytes are converted back to Unicode.

- Hawk does not yet support Unicode internally. The Windows API function `WideCharToMultiByte()` is used to convert Unicode to ANSI Code Page characters when a file is loaded and then `MultiByteToWideChar()` is used to convert back to Unicode when the file is written out.
- `Auto-sense file type EOL`: When this box is checked, Hawk will look for line terminations when it first loads a file. It can detect DOS (`cr/lf`), UNIX (`lf`), and Macintosh (`cr`) line terminations. If the file has UNIX-style line terminations, the `E` key is made to insert only a line feed (`lf`). Macintosh translations for both input and output are done at a low level and appear as DOS files to the user.

It is advisable to use the `Auto-sense file type EOL` option when editing UNIX files in Hawk.

- `Suppress reload prompt`: If another process alters a file that you have loaded into Hawk, it will prompt you, asking if you wish to reload the file. If you wish to always reload the file without being prompted, check this box.
 - `Preload file on open`: When checked, this box indicates that all edit files should be completely loaded into virtual memory. The alternative just loads as much as necessary for editing, and retains pointers into the original file. Preloading takes longer, and uses more temporary disk storage, but then you no longer rely on the original file. If you lose your network connection or if the file is somehow deleted, you will still be able to continue.
 - `Preload non-local files`: When checked, this box indicates that you want to load files on non-permanent media completely into virtual memory. Non-permanent media includes floppy disks and network drives.
 - `Auto-validate preloaded files`: When checked, Hawk verifies at critical junctures that the preloaded file you are working on has not been altered by another process.
 - `Auto-validate non-local files`: When checked, Hawk ensures at critical junctures that the file you are working on has not been altered by another process, if that file is on non-permanent media. Non-permanent media includes floppy disks, and network drives. Auto-validating can be a slow process on some remote media. In this case, you may wish to turn off validation.
4. The following options in the `File` and `File System` group can be enabled for quick access to files being loaded in Hawk and as precautionary measures to maintain the integrity of files being edited:
- `Show file list on File menu`: When this box is checked, Hawk maintains a list of up to 9 recently viewed files at the end of the `File` menu. The most recently loaded file appears at the top of the list. To reload one of these files may, select the filename from the list.
 - `Save file on loss of focus`: You may wish to have files saved whenever you switch away from Hawk to another application. If so, check this box. This adds an extra level of security when you are testing programs that might deprive you of the opportunity to save later.

Use file rewrite save method: When this option is off, Hawk uses the traditional method of saving a file. This involves renaming files (write to a temporary filename, rename the old file to the backup file, and then rename the temporary file). Renaming can cause problems on some systems, where special file attributes or links exist. When this option is on, Hawk avoids these problems by working on the original file, and making a copy of it when it creates the backup file.



The **File Rewrite Save Method** is also recommended for saving large files; refer to [Chapter 14 Working with Large Files](#) for more information.

File Backups and Auto-save

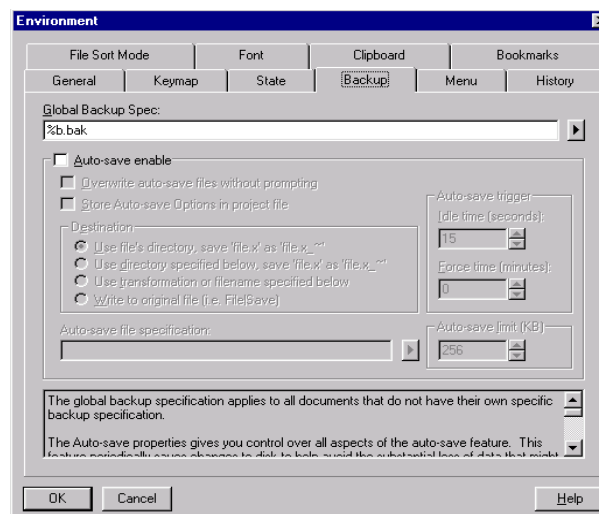
This section discusses how to set up file backup and auto-save options in Hawk. The options can be set globally, for all files being edited, per file type (for instance, .CPP, .C, .TXT), or per individual file.

Global Backup and Auto-Save Settings

The global backup and auto-save settings can be made on the Backup tab of the Customize -> Environment dialog. To access this tab:

1. Select the **Customize** menu.
2. Select **Environment**.
3. Click the **Backup** tab.


Figure 13-2. Customize -> Environment -> Backup Tab



4. Address the following fields:
 - **Global Backup Spec:** The global backup specification dictates how the name of a backup file is derived from the original filename. This is the most common method of specifying the name of the backup file, though backup

specifications may be set for each document. The backup specification can indicate the directory in which backups are made, the extension used, and/or formatting strings which transform the filenames into other names.

The default backup specification, `%b.bak` dictates that the backup file be made in the same directory as the original file, but bearing the extension `.BAK`.

Press the arrow  at the end of the `Global Backup Spec` field for a drop-down list of additional specification options; click an item in the list to automatically insert it in this field.

The backup specification is only used when backups are enabled. Refer to the topic [Formatting the Backup Specification](#), in this chapter, for more information.

- `Auto-save Enable`: Check this box to access options related to the auto-save feature. This feature periodically saves changes to disk to avoid the substantial loss of data that could otherwise result from a power outage or other minor catastrophe. Normally these saves are performed using a filename other than the original. This allows you to determine when you are ready to overwrite the original file. You can have the auto-save feature make saves based on elapsed time, or based on keyboard inactivity, or both.
- `Overwrite auto-save files without prompting`: If Auto-save detects a file of the same name as it wants to use, it checks to see if it is a file that it created during the current session. If it is not, Hawk prompts you before overwriting. The file may not be an auto-save file at all, or it may be an auto-save file from a session that terminated abnormally. In these cases, you may wish to retain the file. On the downside, if you are not present to respond to the prompt, the file will not be auto-saved. This check box allows you to specify that the file be overwritten regardless of its origin without prompting.
- `Store Auto-save Options in Project File`: Check this box to store auto-save settings with the project that is currently open. If the box is checked but no project is open, the selected options are stored as default settings for use with future projects.
- The `Destination` section offers four options for controlling the directory and file-extension to be used for auto-saved files. Descriptions of the options are as follows:
 - `Use file's directory, save file.x as 'file.x_~'`: Auto-saved files are saved to the same directory as the file being edited. A tilde is used as the third character of the extension. If the file being edited has only one character in its extension, an underscore is used as the second character of the auto-save filename.
 - `Use directory specified below, save file.x as 'file.x_~'`: Auto-saved files are saved to the directory specified in the Auto-save Directory field. A tilde is used as the third character of the extension. If the file being edited has only one character in its extension, an underscore is used as the second character of the auto-save filename.

- `Use transformation or filename specified below`: Auto-saved files are saved with the directory and filename or equivalent transformation pattern specified in the Auto-save file specification field. (The field is only enabled when this box is checked.)
- `Write to original file (i.e. File -> Save)`: Auto-saved files are saved to the file currently being edited.
- `Auto-save Directory`: Enter here the name of the directory, if you want auto-save files created in a central location. If you don't name a directory for auto-save files, they will be created in the same directory as the original file.
The filename used will be the same as that used on the original file, except that the third character of the extension will be a tilde (~). If the extension is less than two characters, it will be filled with underscores (_). (For more details, see the topic [Transformation Patterns](#), in this chapter.)

Examples

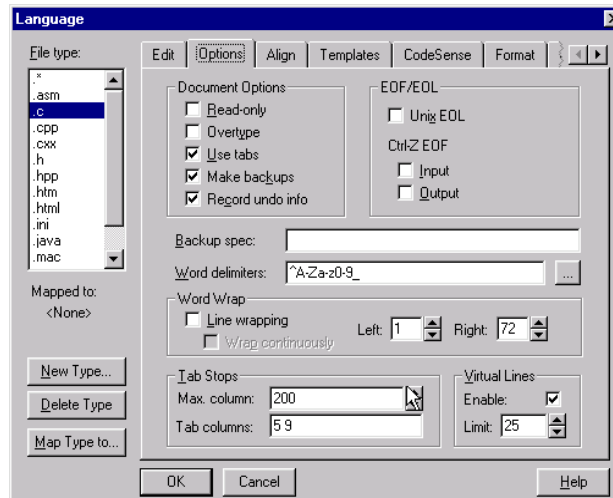
- The file `FOO.C` would be auto-saved under the name `FOO.C_~`.
- A file whose name already has three characters, such as `FOO.PAS`, is auto-saved under the name `FOO.PA~`. In this case the `S` was replaced with a tilde.
- A file with no extension, such as `README`, would be saved with the extension `._~` (the new name would be `README._~`).
- The `Auto-save limit` list box is used to limit the size of files that are auto-saved. Files that exceed the specified size will not be auto-saved.
- `Force time`: This entry sets the absolute time interval that is allowed to pass without saving to disk. This type of auto-save is performed without regard to keyboard or other activity. Enter the number of minutes you want to allow before auto-save interrupts editing to save the file. A value of zero has the effect of turning this kind of auto-save off.
- `Idle time`: This entry sets the interval of inactivity that will trigger an auto save. Inactivity means no key has been pressed and no button has been pushed. Enter here the number of seconds of inactivity you want to allow. A value of zero has the effect of turning this kind of auto-save off.

Backup Settings for Specific File Types

To make a backup specification for a specific file type (that overrides any global specification), click the `Options` tab on the `Language` dialog:

1. Select `Customize`.
2. Select `Language`.
3. Click the `Options` tab.

Figure 13-3. Customize -> Language -> Options Dialog



4. In the `File type` box, highlight the extension of the desired file type.
5. In the `Backup spec` field, enter a string that dictates how the name of a backup file should be derived from the original filename. The default backup specification, `%b.bak` dictates that the backup file be made in the same directory as the original file, but bearing the extension `.BAK`. The backup specification is only used when backups are enabled (the `Make backups` box is checked).



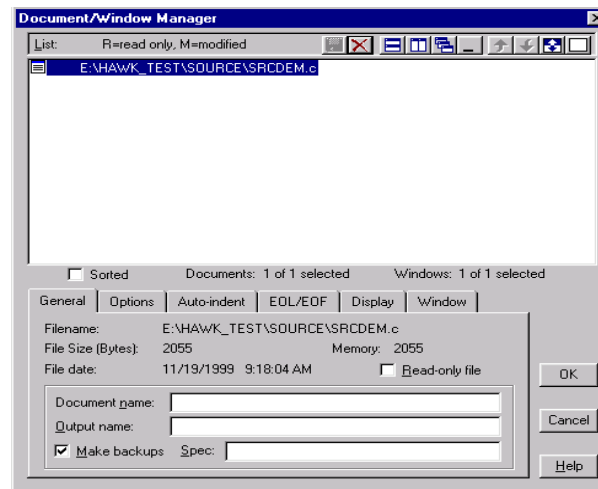
Refer to the topic [Formatting the Backup Specification](#), in this chapter, for more information.

Backup Settings for Individual Files

To make a backup specification for an individual file (that overrides any global or file type specifications), click the **Details** tab of the Document/Window Manager dialog.

1. Select **Document** -> **Manager**.
2. Click the **General** tab.

Figure 13-4. Document -> Manager -> General Dialog



3. In the Document List box, select the document whose settings you want to edit.
4. Check the **Make backups** check box.
5. In the Spec field, enter a string that dictates how the name of a backup file should be derived from the original filename. The default backup specification, %b .bak dictates that the backup file be made in the same directory as the original file, but bearing the extension .BAK.



Refer to the following topic, [Formatting the Backup Specification](#), for more information.

Formatting the Backup Specification

Hawk allows you to specify where and under what name you wish to store backup files. It even allows you to define how to derive the root or extension of the backup filename from the file being backed up.

Backup locations and the filenames used are controlled with formatting strings.

Formatting strings contain format controls and transformation patterns that tell Hawk how to deal with file names that are as yet unknown. Format controls begin with a single percent sign (%). Any other text is treated literally..



It is possible to specify a format that will result in an illegal filename. Hawk does not attempt to ensure that the resulting filename is legal. When you attempt to backup the file, an error will occur, just as if you had specified an illegal filename for saving a file.

Leaving the formatting strings empty at both the global and local levels effectively turns backups off.

The global string is located on the Customize -> Environment -> Backup dialog; the local strings are located on the Customize -> Language -> Options and Document -> Manager -> General dialogs.

Format Controls

The format controls available for use in backup formatting strings are listed in the table below. Note that optional portions of these format controls are enclosed in italicized square brackets. Examples are based on the output filename `C:\SRC\FOO.BAR`:

Table 13-1. Backup String Format Controls

Control	Represents	Description
<code>%v</code>	volume	The drive letter and the colon. (Example, <code>C:</code>)
<code>%p</code>	path	Does not include the drive (volume) or filename. Always ends with a backslash. (Example: <code>\SRC\</code>)
<code>%r</code>	root	Does not include extension, drive (volume) or path. May contain an optional transformation pattern, as described below. (Example, <code>FOO</code> if no transformation pattern supplied)
<code>%d</code>	directory	Includes the path, less the drive (volume) specifier and filename, but ends with a backslash only if describing the root directory. (Example: <code>\SRC</code>)
<code>%e</code>	extension	Includes the dot that separates the basename from the extension, unless the name of the output file has no extension. May optionally contain a transformation pattern, as described below. (Example: <code>.BAR</code> if no transformation pattern supplied)
<code>%b</code>	basename	The entire name of the output file (fully qualified), less the extension. (Example: <code>C:\SRC\FOO</code>)
<code>%{??~ __}e</code>	modified extension	
<code>%x</code>	project path	
<code>%y</code>	project root	
<code>%%</code>	percent	A single percent sign is represented by two consecutive percent signs (<code>%%</code>).
<code>\$(name)</code>	macro expansion	

Example

The formatting string which represents the initial default is `%b.bak`. You would set this default using a function call like the one below:

```
BufSetGlobalBackupSpec="%b.bak"
```

Transformation Patterns

The root and extension format controls, described above, may contain transformation patterns. These patterns describe modifications within the root filename and extension. You enclose the pattern within curly braces and place it between the percent sign and the control character, `r` or `e`, respectively.

You supply the pattern in two parts, the override pattern and the fill pattern:

- The override pattern specifies characters, each of which replaces characters at the same position within the original root or extension string. The override pattern also may limit the length of the resulting string.
- The fill pattern specifies characters to fill empty positions within the root or extension string.

A single vertical bar (|) separates the override pattern from the fill pattern. If you are supplying only the override pattern, you may omit the vertical bar. Supplying only the fill pattern results in an empty string.

The rules are the same for using transformation patterns, whether you are using them on the root of a filename or its extension. There is one noteworthy difference in their result, however. After the transformation is performed on an extension, the resulting extension is examined. If the extension is not null, a dot is added to the beginning of the extension. This is true whether or not the original extension was null.

Override Pattern

There is a character in the override pattern for each character in the resulting string. The character at each position may be either a valid filename character, or a question mark (?).

- If a filename character is specified, the character in the original string is replaced by the specified character.
- If a question mark appears at a given position in the string, the character in the original string is used.
- If no character appears in the override pattern for a given position (example, the root is shorter than 8 characters or the extension is shorter than 3), the resulting string is truncated at that position, even if you have supplied a fill pattern.

Fill Pattern

The fill pattern may contain only valid filename characters. The character at a given position in the pattern is used in the resulting string if that position in the original string is empty (for example, the original string is shorter than the fill pattern) and a question mark is specified by the override pattern.

The following examples demonstrate the effects of various override and fill patterns:

Table 13-2. Transformation Patterns

Pattern	Original Name	Backup Name
%r%{??~ ___}e	FOO.CPP	FOO.CP~
	FOO.C	FOO.C_~
%r%{??~}e	FOO.CPP	FOO.CP~
	FOO.C	FOO.C

Table 13-2. Transformation Patterns

Pattern	Original Name	Backup Name
%r%{~??}e	FOO.CPP	FOO.~PP
	FOO.C	FOO.~
%r%{~}e	FOO.CPP	FOO.~
	FOO.C	FOO.~
%r%{?}e	FOO.CPP	FOO.C
	FOO.C	FOO.C
%{???????~ ~~~~~}r %e	FOO	FOO~~~~~
	FOO.C	FOO~~~~~.C
	TESTFILE.C	TESTFIL~.C
%{?????}r%{??? bak}e	FOO	FOO.BAK
	FOO.C	FOO.CAK
	TESTFILE.C	TESTF.CAK

Making Files Read-only

Files can be assigned a read-only (non-editable) status at several locations within Hawk, much like backup specifications. The hierarchy of these locations is indicated below, and details of each location follow.

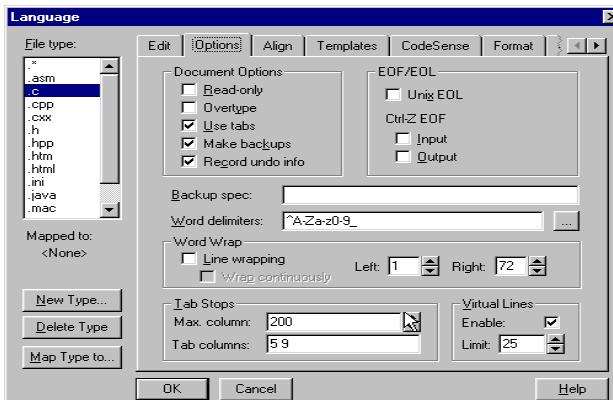
- **Customize -> Language -> Options** dialog – Allows you to make all files of a specified type read-only (For example, you could make all .INI files read-only).
- **Document -> Manager -> General** – Allows you to specify that an individual file will be read-only, overriding the setting for its file type.
- **File -> Open** – Directs Hawk to open this file as read-only, overriding all prior settings.

File Types

To make files of a certain type read-only, select the **Options** tab of the Language dialog and complete the following steps:

1. Select the **Customize -> Language -> Options**.

Figure 13-5. Customize -> Language -> Options Tab



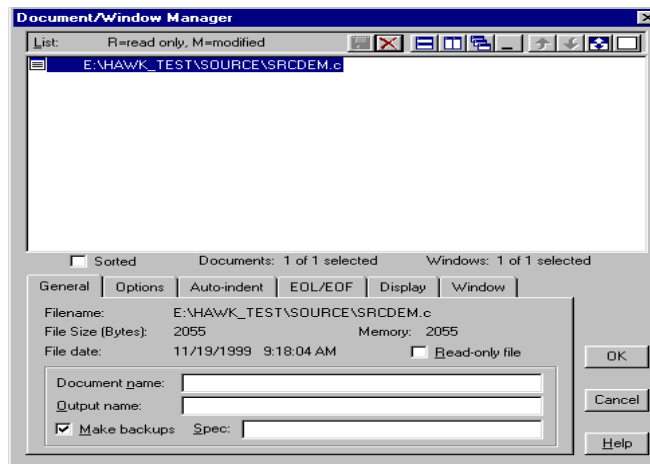
2. In the `File type` box, select the extension for the desired file type.
3. In the `Document Options` group, select the `Read-only` check box. Files of the specified type may not be edited, unless this setting is overridden on the `Document -> Manager -> General` dialog, or the `File -> Open` dialog.

Individual Files

To make an individual file read-only, select the `General` tab on the `Document -> Manager` dialog and complete the following:

1. Select the `Document -> Manager -> General`

Figure 13-6. Document -> Manager -> General Tab



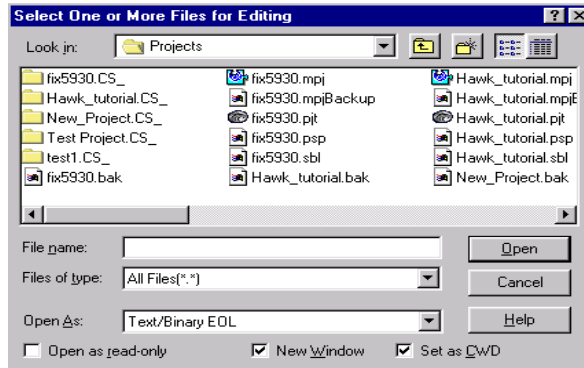
2. Select a document in the `Document List` field to edit its settings.
3. Check the `Read-only file` check box to prevent editing of this file.

Individual File upon Opening

To make a file read-only upon opening, complete the following steps:

1. Select the **File** menu.
2. Select **Open**.

Figure 13-7. File -> Open Dialog



3. Browse to find the file you want to open and select the filename.
4. Select the **Open as read-only** check box.



14

Working with Large Files



Hawk can accommodate files as large as two gigabytes.

There are several ways to configure Hawk to make it more stable and efficient when working with large files, including the following:

[Swap Blocks](#)

[Backup Files](#)

[Scroll Bars](#)

[Preloading Files](#)

[Turning off ChromaCoding](#)

[Saving Large Files](#)

Swap Blocks

Hawk uses swap blocks to manage the memory allocated for an open, active file. Hawk is fairly conservative about the amount of memory it initially allocates (100 8K blocks). When all of the swap blocks initially allocated are used up, Hawk starts swapping to disk. If you regularly work with large files, and you have plenty of memory available, you should consider increasing the number of swap blocks used by Hawk.

Example

The 32-bit default for swap blocks is 100 8K swap blocks. You may want to try increasing this (the maximum is 250).

How to Change the Number of Swap Blocks

Set the number of swap blocks in your configuration file (`MWHAWK.INI`) located in the Hawk home directory. Find the `[Editor]` section of this file and insert a line under that heading similar to the one shown below, then restart Hawk for the changes to take effect:

```
[Editor]
SysSwapBlocks=200
```

Block Size

The size of the swap block (discussed above) can be increased using the function:

```
-BlockSize=
```

When the size of the block is increased, fewer blocks are needed, expanding Hawk's capacity for handling longer lines and larger files.

The default block size is 0x2000 (8192) but the range can be from 0x1000 to 0xf000 (4096 to 61440 decimal).

- The default block size (0x2000) allows a single 500-Mb file to be opened and edited in Hawk.
- Increasing the block size to 0x8000 will allow a single 2G file to be opened and edited, or two 1G files.
- Increasing the block size more will allow additional large files to be loaded, but 2G will still be the individual limit.

As block sizes increase, the speed of loading large files may be diminished, but overall speed should increase.

The `-BlockSize=` parameter can be used from the Hawk shortcut command line, a DOS command line, or any of the `sync` program command lines.

To modify the command line for the shortcut:

1. Right-click on the Hawk shortcut.
2. Choose **Properties** from the resulting pop-up menu.
3. Click the **Shortcut** tab.

- In the Target edit box, add `-BlockSize=0x8000` after the CW executable.

Example: `C:\cw\cw32.exe -BlockSize=0x8000`

Consider Your Resources

You should consider how much memory is available on your system and how big your block size is before changing the number of swap blocks. You will know that you have made the number too high if you begin to see degradation in the performance of Windows itself.

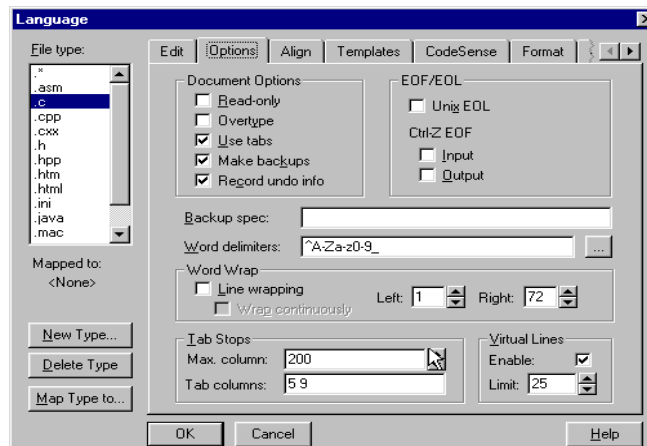
Backup Files

You may wish to have Hawk not make backup copies when working with very large files. In most cases this can be both a waste of time and disk space.

To turn off backups for a specific file type, complete the following steps:

- Choose the **Customize** -> **Language** dialog.
- Click the **Options** tab.

Figure 14-1. Customize -> Language -> Options Dialog



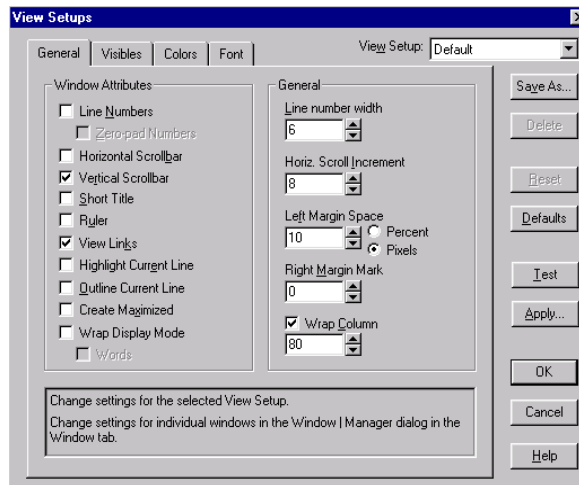
- In the **File type:** box, select the file type(s) for which you want to turn off backups.
- Clear the check box labeled **Make backups** in the **Document Options** group.

Scroll Bars

To edit large files as soon as you load them, turn off scroll bars. It is too late to turn off scroll bars after you have loaded the file, so you will want to make **No Scrollbars** the default. To do this, complete the following steps:

- Select **Customize** -> **View Setups**.
- In the **View Setups** dialog, click the **General** tab.

Figure 14-2. Customize -> View Setups -> General Dialog



3. In the Window Attributes group, make sure that the Vertical Scrollbar and Horizontal Scrollbar check boxes are not checked.
4. Click **OK**.

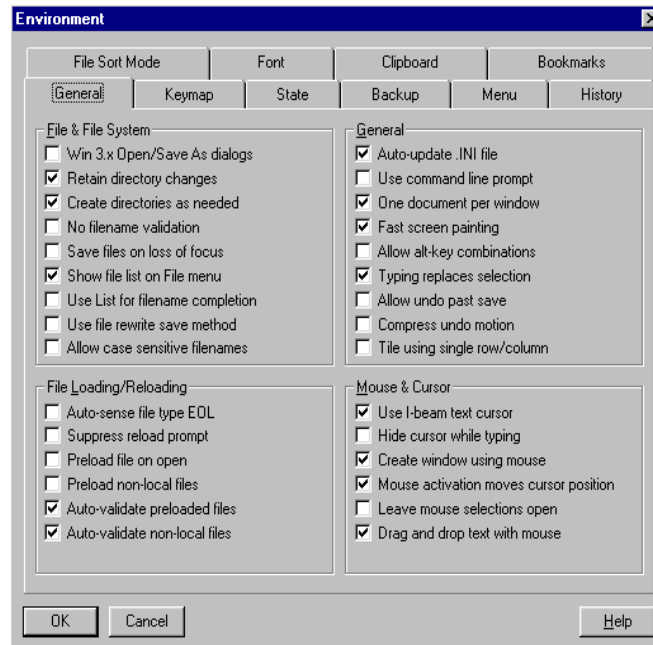
Preloading Files

When Hawk preloads files, it loads the file completely into virtual memory, which can slow things down if the file being loaded is fairly large. There are two options that enable the Hawk feature for preloading files. Make sure the options are not marked if you want Hawk to handle large files more efficiently.

To disable the preloading options:

1. Select the **Customize** -> **Environment** dialog.
2. Click the **General** tab.

Figure 14-3. Customize -> Environment -> General Dialog



3. In the File Loading/Reloading section, make sure the Preload file on open and Preload non-local files on open check boxes are not checked.

When the options are turned off, instead of loading the whole file into memory, Hawk only loads as much of the file as necessary for editing and maintains pointers to the original file. See [Chapter 13 File Loading and Backup](#) for more information.

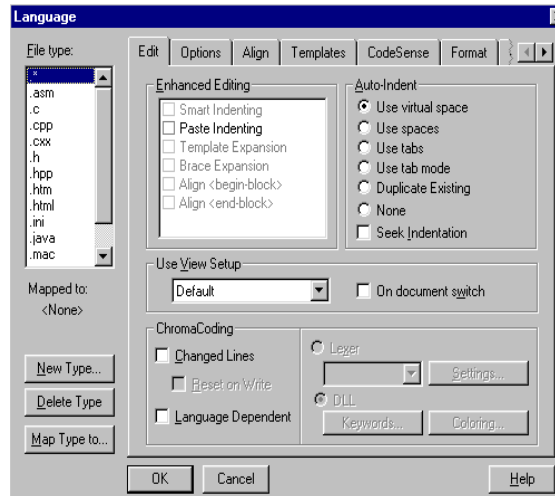
Turning off ChromaCoding

If speed is of the utmost importance when loading large files, you could optionally turn off the Hawk ChromaCoding feature. This feature colors programming language syntax structures.

To turn off ChromaCoding, complete the following steps:

1. Select the **Customize -> Language** dialog.
2. Select the file type being edited in the list of **File Types** on the left.
3. Click the **Coloring** tab.

Figure 14-4. Customize -> Language -> Coloring Dialog



4. Uncheck the following features in the ChromaCoding section:
 - **Changed Lines**: This causes Hawk to color lines that have been changed.
 - **Language Dependent**: This causes Hawk to color files according to the syntax of a particular programming language.

Saving Large Files

The Hawk default file saving method is not the preferred method for saving large files. For large files, use the **File Rewrite Save Method**. Both saving methods are described in this section.

Default File Saving Method

The default method Hawk uses to open and save files works in the following way:

Given a file, `FOO.C`

- Open `FOO.C` into Hawk.
- Hawk creates `FOO.000` as a temporary file.
(The `.000` extension will be `.001` if `.000` already exists, `.002` if `.000` and `.001` already exists, and so on.)
- If Hawk is set to make backups, `FOO.C` is renamed to `FOO.BAK`. If the option to make backups is not turned on, `FOO.C` is deleted.
- When the file is saved, `FOO.000` is renamed to `FOO.C`.

File Rewrite Save Method (for Files over 1MB)

The **File Rewrite Save Method** is simpler than the default saving method. It works in the following way:

Given a file, `FOO.C`

- Open `FOO.C` into Hawk.
- If backups are turned on, `FOO.C` is copied to `FOO.BAK`. If not, nothing happens.
- When the file is saved, the changes are written back to the original file.

The `File Rewrite Save Method` offers the following advantages when saving large files:

- Space normally needed for the temporary file (`FOO.OOO`) is no longer required. This minimizes the amount of memory needed when opening the file.
- Since the `File Rewrite Save Method` only writes out the changes made to the file (as opposed to rewriting the whole file), the portions of the file that have not been modified will not be rewritten, reducing the time needed to save the file.

To turn on the `File Rewrite Save Method`, complete the following steps:

1. Select the `Customize -> Environment` dialog.
2. Choose the `General` tab.
3. In the `File` and `File System` section, put a check mark in the `Use file rewrite save method` option.



15

Extending Hawk

The Hawk tools provide a means for developing the necessary functions for handling specific tasks. The tools can be used individually or together.

The following sections are included in this chapter:

[Hawk API](#)

[Macros and Macro Languages](#)

[Perl](#)

[AppBasic](#)

[API \(C-like\) Macros](#)

[Making DLL Add-Ons](#)

[Core Services](#)

[Keyboard Command Sets](#)

[Supplemental Language Support](#)

[Auxiliary Services](#)

[Sample DLL](#)

[Dissecting a Hawk DLL](#)

[Making Changes and Additions](#)

[Creating New Keymap Command Sets](#)

[Recompiling a DLL](#)

Hawk API

Hawk has many API functions that can be used interactively from within the interface or attached to menu items, buttons, or keystrokes. APIs can also be used in custom macros and DLLs designed for extending Hawk.

This chapter describes using Hawk APIs from the `Command Key`, macros, and DLLs. Help and examples for most Hawk APIs can be accessed in Hawk's online help, using the `Functions` or `Search For Help On` menu items under the `Help` pull-down menu.

Using the API from the Command Key

This section describes using the `Command Key`.



Remember that the `Command Key` is `F9` if you are using the CUA or the `vi` keymap command set, and `F10` if you are using the `BRIEF`-compatible keymap command set. In keymaps where there is no `Key Command` assignment, selecting `Tools -> API Command` always accesses the prompt.

To use the Hawk API, you must know the name of the particular function you want to execute. Hawk's online help describes the Hawk API and specific functions. Use the `Functions` option on Hawk's `Help` pull-down menu to find a list of some of the API functions available, or use the `Search for Help On` item to do a search for APIs that might appear under a particular topic.

Displaying Return Values With the Command Key

Hawk APIs that have return values are much more useful if the return value can be accessed or viewed. To get the `Command Key` prompt to display the return value of a function, begin the command with a question mark.

Following is an example of such a command, prior to sending the command off for processing:

```
?BufQModifiedCount
```

When the return type is a numeric value, Hawk displays that number on the status line in both decimal and hexadecimal notation. If the return type is a pointer to a string, Hawk displays the contents of that string.

Examples of Command Key Usage

```
BufSetHexBinary
BufSetHexBinary()
bufsethexbinary
```

All three of the above examples do the same thing. They put Hawk in the `Hex Editing` mode, with the cursor located in the "binary" portion of the display.

```
MovNextChar( 2 )
Movnextchar 2
```

The above examples advance the cursor position by two characters.

```
BufEditFile( "c:\projecta\debug.c" )
```

```
bufEditFile "c:\projecta\debug.c"
```

```
BufEditfile=c:\projecta\debug.c
```

Either of the above commands opens a document for C:\PROJECTA\DEBUG.C.

```
kmapAssign( "<ctrl-a>", "BufSetHexAscii" )
```

Assigns the function BufSetHexAscii to the keystroke Ca.

```
MarkBeginSel( SELECTION_INCLUSIVE )
```

Begins an inclusive selection, using one of the labels listed in Hawk's online help.



For more information on Hawk API functions, refer to Hawk's online help.

Command Key Expression Evaluation

The `Command Key` can also do expression evaluation, which can be useful for programming, debugging, or simply as a calculator. With expression evaluating, a numeric expression that uses C language operators and grouping, can be evaluated, or calculated, from within the `Command Key` prompt and the `Command Key` will in turn display the results.

To use the `Command Key` for expression evaluating, signify that the expression is one for evaluation by preceding the expression with two question marks.

Hawk identifiers can be used in these expressions. Here are examples of expressions to be evaluated:

```
??SELECTION_INCLUSIVE
```

```
??0xcf54-3*4
```

```
??MARK_GLOBAL+1
```

The following operators are supported in expressions, grouped by precedence, and listed in order of decreasing precedence:

Table 15-1. Supported Operators

Operator	Operation
()	Grouping
-	Unary Minus
~	Bitwise Complement
!	Logical NOT
/	Division
*	Multiplication
%	Modulus

Table 15-1. Supported Operators (Continued)

-	Subtraction
+	Addition
<<	Bitwise Shift Left
>>	Bitwise Shift Right
>	Greater Than
<	Less Than
<=	Less Than or Equal
>=	Greater Than or Equal
=	Equivalence
!=	Non-Equivalence
&	Bitwise AND
^	Bitwise Exclusive OR
	Bitwise OR
&&	Logical AND
	Logical OR

Macros and Macro Languages

A macro can be defined as something simple that represents something bigger or more complex. Under this definition, macros in the Hawk environment include everything from keystroke recordings that are assigned to keys, and the % macros Hawk uses in command lines, up to sophisticated code written in a programming language.

A Macro Language provides a method for creating macro source code, which may contain control structures and variables, and which is normally interpreted at runtime. Hawk has three macro languages from which to choose.

Each of the macro languages used for extending Hawk has its strengths. [Table 15-2](#) describes some pros and cons that appear in the three languages. The languages

themselves are discussed in detail in the sections on [Perl](#), [AppBasic](#), and [API \(C-like\) Macros](#).

Table 15-2. Macro Languages

Macro Language	Strengths and Weaknesses
Perl Macros	<p>Perl is, along with JavaScript, perhaps the most popular language for writing extensions to web pages. The syntax is similar to C or AWK, but it has many features for the Internet built-in.</p> <p>This language is least suitable for writing simple functions for assignment to keys, because of the time it takes for the interpreter to be loaded initially. It does, however, allow Perl programmers to program in a familiar language, using familiar extensions and libraries. Hawk's version of Perl is based on the Gnu-released Perl interpreter.</p>
AppBasic Macros	<p>AppBasic is the Hawk macro language that is similar to Microsoft's Visual Basic for Applications. It has its own editor, debugger, and basic functions. You'll find it on a tab of the Output Window.</p> <p>Using this macro language, you can access functions in the following locations:</p> <ul style="list-style-type: none"> • The Microsoft Windows API. • All of the Hawk API functions, except those designated as non-interactive. <p>Functions in most any external DLL.</p> <p>For those who are familiar with basic in any of its various forms, this is an attractive option for both simple and complex macros.</p>
API (C-like) Macros	<p>The Hawk API Macro language is the simplest of the available macro languages. Using C-like structure and syntax, it allows you to quickly create functions suitable for assigning to keys and other simple uses. When writing an API Macro, you can use any function that is available from the API command line.</p> <p>You can write API Macros in the <code>TOOLS -> API Macros</code> dialog, or you can write them in one of Hawk's standard edit windows, and then copy them into the API Macros dialog.</p>

DLL Extensions

You can write extensions for Hawk using any compiler that can produce DLLs. This capability is referred to as `DLL Extensibility`. Most sophisticated extensions are written using `DLL Extensibility`. This method of extending Hawk has the following benefits:

- familiarity in compiling and debugging tools
- speed

- access to functions in other DLLs and libraries



For details on [DLL Extensibility](#), refer to the [Making DLL Add-Ons](#) section.

- Hawk DLLs, or Add-Ons, can be loaded interactively from the **Tools -> API Command** dialog.

DLL Extensibility, however, does have some limitations, including the following:

- It is inconvenient for simple jobs or single use programs.
- It provides little protection against user error, which could cause a system crash.
- It needs its own, separate compiler.

Under these circumstances, you may want to consider extending Hawk using one of the supplied macro languages (Perl, AppBasic, and API).

Where it is Defined

Once an extension has been created, there is an opportunity for confusion as to exactly which function being executed is coming from. It could be a Hawk built-in function, a Hawk DLL function, an AppBasic macro, a Perl sub, a keystroke macro or an API macro. The following command helps to resolve the ambiguity.

```
LibFunctionExistsWhere()
```

This command returns a string identifying the location of the function name that is supplied as its argument. Listed below are a few sample responses.

Table 15-3. Function Responses

Function	Response
BufQCurrentLine	Built-Ins:BufQCurrentLine
DlgPrint	cwdialog:DlgPrint
MyPerlFunc	cwPerli:myPerl!MyPerlFunc
_jav_init	_jav_init->cwstart:_java_init

The last example shows the response where a replacement function exists. The Perl example shows that both the Perl script filename and the sub name are given.

For non-LibExported functions, such as those connected by responding to `EVENT_LIB_EXISTS_FAILED` and `EVENT_LIB_EXEC_FAILED`, the 'where' string is supplied by the responder to a new event `EVENT_LIB_EXISTS_WHERE`. The event handler supplies a 'where' string (allocated) as the return value or zero if it does not claim the function.

Perl

There are two useful ways of looking at a Perl script. One is to look at Perl as an entity that may be executed in its entirety to perform the task specified by the operations of the script. Another way of looking at the script is as a collection of macros, each subroutine being a macro in its own right. Using Perl with Hawk, you can make use of Hawk custom facilities that reinforce these views as you have need.s



For more information see the [Loading and Running Scripts](#) section.

Getting Started with Perl

The following two loadable DLLs are necessary to use Perl with Hawk:

- Perl interpreter DLL
- Perl language support DLL

To enable these modules, go to **Customize** -> **Libraries** and select the check box for **Perl Extension Language Interpreter**, and also **Perl**.

If the **Output Window** is not already showing, select it from the window menu. You should then see a **Perl** tab on the **Output Window**.

- Even if you have Perl already installed on your system, you must check the **Perl Extension Language** check box to enable the **Perl** tab for the **Output Window**.

The **CWP.DLL** is a Perl extension module that provides access to Hawk API functions from within Perl scripts. In general, you can use any of the Hawk API functions listed in the online help, except for those whose data types are incompatible with Perl. For a list of available Hawk API functions, see the contents of the **CWP.PM** file.

Creating and Editing Perl Scripts

Perl scripts can be created and edited in standard Hawk edit windows. There are some sample Perl script source files in Hawk's **MACROS** subdirectory. Load a few of these and have a look.

Perl scripts are usually packaged in text files bearing a **.PL** extension. A Perl script consists of a 'main' section, and zero or more subroutine definitions. The main section is defined as any code not contained within subroutine definitions.

Example

The following Perl macro contains everything in its main section:

```
use CWP;
$filename = CWP::BufQFilename;

$filespec = "%p%r%e";
```

```
$newname = CWP::TransformFilename($filename, $filespec);
$newname =~ tr/\\\/\//;

$newname =~ tr/A-Z/a-z/;
CWP::LibFunctionExec "SetStringMacro(UNIXFILE, $newname, 0)";
```

Perl Window

The Perl tab on the Output Window acts as a virtual console for the Perl interpreter. That is, it replaces the `stdin`, `stdout`, and `stderr` devices. You can scroll through the output of Perl Scripts in this window. There is a limit on how many lines of output will be retained in this window. The default is 100 lines. This setting is in the Tools -> Perl Macros -> Properties dialog.

To help distinguish `stdin`, `stdout`, and `stderr` elements of the Output Window, CWPerl uses three different colors.

- `stdout` is displayed using the output color.
- `stdin` is displayed using the color defined for line numbers.
- `stderr` is displayed using the color designated for comments.

You can find the settings for these colors on the Customize -> View Setups -> Colors dialog.

Pop-up Menu and Options

When you right-click on the Perl tab, or any portion of the Perl Window portion of the Output Window, a menu pops up that has a number of additional options. If you don't find what you are looking for on the toolbar, or the Tools menu, look in this menu.

The Properties item on the pop-up menu brings up a dialog that lets you control how Perl interacts with Hawk. This is where you specify options you would otherwise provide to Perl on its command line, when invoking it from the shell prompt.

There are a variety of choices from which to select Perl's input source and output destination including the Perl Window, the current buffer, the clipboard, the current scrap buffer, and the current selection.

Different combinations of source and destination can produce interesting effects.

Example

If Current Document is selected for both source and destination, a Perl script will see the entire content of the current document as its `stdin` and the buffer will be replaced with Perl's `stdout`.

You can accomplish a similar operation on the contents of a selection choosing Selection for both source and destination.

Input source and output destination can also be set from within a Perl Script. See the file `UPCASE.PL` in the `MACROS` subdirectory of the Hawk installation directory for an example of how this is done.

Online Help

The Perl Manual by Larry Wall is supplied in online form, in the help file `Perl.HLP`. The topics in this file are part of Hawk's standard help system. This means that you can get Perl function help by placing the cursor on the name of the function and pressing **C!**. You can also press **!** with the cursor in the `Perl` window to get other Perl help.

Loading and Running Scripts

There are two different ways to run a Perl script in the `Perl` tab of the `Output Window`. You can load the script directly on an as-needed basis, or you can load a Perl macro and run it any time you wish.

Running a Script Directly

To run a Perl script directly, type the following command from the Hawk API Command Line (Command Key/ Tools -> API Command):

```
PerlExec <script file> <any parameters>
```

Where `<script file>` is the fully qualified path and file name of the script file, and `<any parameters>` is any parameters required by the script file.

This will load the script, execute it, and unload the script automatically.

Loading a Perl macro

Loading a Perl macro is much the same as running a Perl script directly, with the added advantage of being able to invoke the subroutines of the script individually by simply typing them at the command line.

There are three ways to load a macro:

- Go to the `Tools` menu and select `Perl Macros...` and then select `Load macros`.
 - Select supplied Perl macros by checking the appropriate box in the top of the dialog in the area called `Hawk macros`.
- Any Perl macros stored in the `CW32\MACROS` directory will show up in the Hawk macros list.
 - Add user-defined macros from here by pressing the `Add` button.
- Another way to load Perl macros is by right-clicking the `Perl` tab in the `Output Window` and selecting `Interpreters`. This brings up the `Hawk Perl interpreters` dialog. Load the macro by pressing the `Load...` key. All subroutines for the loaded macro (including `<main>`) are automatically displayed when you select a macro from the list. You may select an individual subroutine from the list to run.
- Use the Hawk API command:

```
PerlLoad <filename>
```

Once the macro is loaded, you can invoke it by going to the Hawk API Command Line and entering the name of the subroutine. As previously mentioned, you may also run the subroutine as follows:

1. Right-click the Perl tab and select **Interpreters**.
2. Select the macro from the list.
3. When the subroutine list is displayed, select the desired subroutine.
4. Press the **Run** button.
5. You will be prompted for parameters; you may enter them if necessary, or simply press the **ENTER** key to run without parameters.

Executing a Script

You can load and execute a Perl script from the Hawk API Command Line by using the following command:

```
PerlExec filename.pl
```

If you only want to execute a specific subroutine in a script use the following:

```
PerlExecSub filename.pl subname
```

In both of cases, you can include additional 'command line' parameters. All text following the syntactical elements shown in the two commands is broken down into 'parameters' using Hawk's normal command line parsing rules, with respect to spaces, quotes, escapes, etc.

To execute a simple Perl script that can be typed in one line, use the following command:

```
PerlExecStr 'Perl-string'
```

Interpreters

Perl for Hawk is capable of hosting multiple simultaneous interpreters. The three commands create an interpreter, parse the input, execute the script and then dispose of the interpreter. You may alternately load one or more interpreters by using the command:

```
PerlLoad filename.pl
```

This command also may be supplied command line arguments although they will only be utilized to the extent that they affect loading the interpreter. This command can be run several times to load different scripts. From time to time, a loaded interpreter may be utilized by the command:

```
PerlRun filename.pl
```

If this command is issued as shown, the 'main' of the script will be executed. Alternately, you can give a subroutine name as follows:

```
PerlRun filename.pl subname
```

As with previous commands, parameters may be supplied following either form, however, the first form will need an extra empty string ("") to represent the 'main' function.

Accessing Hawk Functions from Perl Scripts

More than 800 Hawk functions can be directly accessed from within Perl scripts in Hawk. To do so, you'll need to place the following line near the top of your script:

```
use CWP;
```

This tells Perl to load `CWP.PM`, which contains descriptions of functions contained in the compiled Perl extension module `CWP.DLL`. The `CWP.DLL` consists of small C functions that call the Hawk functions after first preparing the parameters and then make the return value, if any, available to Perl.

You'll need to use the `CWP::` prefix on Hawk functions to reference them, unless you take further steps (described later). This prefix tells Perl that the following function name is present in the `CWP` module. As an example, to find out where the cursor is on in the current document, you could use Hawk's `BufQCurrentLine` API in the following manner:

```
$line = CWP::BufQCurrentLine();
```

Constants that are held in Hawk's lookup table can be accessed using a special function. For example, to see if there is a column selection present you could use the following:

```
if (CWP::MarkQSelType() ==
    CWP::CWConst("SELECTION_COLUMN"))
{
}
```

A special function is also provided to execute any built-in or any function made available through `LibExport`. This function is similar to the Hawk API function `LibFunctionExec`. An example follows:

```
CWP::CWExec("ConfigFileRead", "[Editor]", 0);
```

This is equivalent to:

```
CWP::LibFunctionExec("ConfigFileRead [Editor] 0");
```

The primary difference is that in the latter example, the string will be parsed to separate it into parameters to be passed to the function. In the former, the parameters are explicitly segregated.

Importing Names into the Perl Namespace

If you have Hawk functions that you access frequently and their names do not conflict with standard names, you can import them into the Perl namespace.

To import names into the Perl namespace, modify the `use` command as follows:

```
use CWP ("LibFunctionExec", "CWConst", "BufQCurrentLine");
```

or, equivalently:

```
use CWP qw(LibFunctionExec CWConst BufQCurrentLine);
```

The latter 'quoted word' shorthand obviates the need to type so many quotes. After doing this, the named function may be invoked without the `CWP::` qualifier.

```
$line = BufQCurrentLine();
```

Unloading a Perl macro

When you no longer need an interpreter it can be deleted with the following command:

```
PerlUnload filename.pl
```

Using the Perl Debug Mode

There is a debugger supplied for Perl scripts. The debugger is itself a Perl script (`PERL5DB.PL`). When invoking Perl from a prompt, this script is automatically loaded if you use the `-d` flag. In Hawk, you accomplish the same thing by checking the `Debug Mode` check box in the `Perl Properties` dialog (right-click the `Perl` tab).

When running a script in debug mode, there are three important requirements:

- Use debug mode only when invoking scripts with `PerlExec` on the API command line.
- The script must not already have been loaded by Hawk prior to using the `PerlExec` command (meaning the script cannot be loaded from the `Interpreters` box or by the `Tools -> Perl macros... -> Load macros...` pop-up dialog). If it is loaded, unload it before attempting debug mode.
- Ensure that you have completed executing the script in debug mode before attempting to run the script again.

Accessing Perl Functions

You can also execute the subs of loaded Perl interpreters by typing their names on the Hawk API Command Line. Hawk has a mechanism to allow Add-Ons to respond to indicate their recognition of otherwise unknown (to Hawk) function names. The `AppBasic` interpreter and the Perl interpreter alike respond to these queries.

This means that if you have a Perl script loaded that contains a sub called `MyPerlFunc`, you can execute that function by typing `MyPerlFunc` at the Hawk API Command Line. Perl will check all the loaded Perl scripts to see if at least one of them has the named subroutine.

Avoiding Ambiguity

The same subroutine name may appear in several Perl scripts. Hawk therefore provides a mechanism to specify which Perl script you're referencing.

Example

Suppose that both `SCRIPT1.PL` and `SCRIPT2.PL` have a sub `MyPerlFunc`. You don't know which subroutine will be executed if you simply invoke `MyPerlFunc`.

- One way to avoid this problem is to right-click the `Perl` tab and bring up the `Interpreters` dialog. Select the script you intend to call, and click its subroutine in the subroutine list.
- Another way to specify that you wish `MyPerlFunc` in `SCRIPT2.PL` to be executed, is to invoke it as follows:

```
script2.pl!MyPerlFunc
```

(Note that the extension `.PL` used in this example is optional.)

Without this specific invocation, the loading order usually determines which function will be executed if both an `AppBasic` script and a Perl script contain a given function name.

- Generally, the first macro loaded will be the one to execute.

Special API Functions for Perl

In addition to the functions available from the Hawk API, there are three additional functions available when the Perl Extension Language (CWP) is loaded.

- `DWORD CWConst(LPSTR cw_expr);`

This function provides access to Hawk constants (predefined values), and any "constants" you have defined with `EvalAddStr`, in Perl scripts. For example, you may want to begin a line selection. To do this you would use the constant `SELECTION_LINE` in a call to `MarkBeginSel`.

A more specific example follows:

```
CWP::MarkBeginSel(CWP::CWConst("SELECTION_LINE"));
```

The argument string to `CWConst` can take any form that is acceptable to the Hawk function `EvalExpression`.

- `DWORD CWExec(LPSTR cw_funcname, ...);`

This function provides an alternate form of executing a Hawk function that has been made available via `LibExport()`, such as all functions provided by DLLs, both standard and Add-On. The advantage to using this form, rather than `LibFunctionExec()`, is that you need not compile the arguments that you wish to pass into a string as is required by `LibFunctionExec()`. You could, for example, issue either of the following commands with the same effect:

```
CWP::LibFunctionExec("BufQOffsetEx", $line . " " . $column);
```

OR

```
CWP::CWExec("BufQOffsetEx", $line, $column);
```

- `int CWPerlIO(int mode);`

This function is used to query or set the current Perl I/O mode. You may recall from the `Perl Properties` entry of the Perl popup menu that you can set Perl's input source and output destination using the radio buttons. This function serves the same purpose.

The `mode` argument has two components:

- `Input source`: occupies the least significant 8 bits of the `mode` value.
- `Output destination`: occupies the next most significant 8 bits.

The semantics of the components are as follows:

```
// Perl I/O source/destinations
#define PERL_IN_CON 0x0000 // input from 'console'
#define PERL_IN_BUF 0x0001 // input from current buffer
#define PERL_IN_SEL 0x0002 // input from selection
#define PERL_IN_CLIP 0x0003 // input from clipboard
```

```
#define PERL_IN_SCRAP0x0004 // input from scrap buffer
#define PERL_OUT_CON0x0000 // output to 'console'
#define PERL_OUT_BUF0x0001 // output to current buffer
#define PERL_OUT_SEL0x0002 // output to selection
#define PERL_OUT_CLIP0x0003 // output to clipboard
#define PERL_OUT_SCRAP0x0004 // output to scrap buffer
```

To determine the current I/O mode, simply call `CWPerlIO` with a mode value of -1. In all cases, the return value is the prevailing 'mode' immediately prior to the call. This function is useful to make a Perl script that responds differently depending on certain prevailing conditions.

- The call to `CWPerlIO` should be made before any output operations are performed, since changing the output destination causes all previous output to be discarded.

Files Used by Perl for Hawk

The following files are found in the Hawk home directory after installation:

Table 15-4. Files in the Hawk Home Directory

Filename	Purpose
CWPERL.DLL	Provides language support for Perl source scripts (* .PL).
CWPERLI.DLL	Adds the Perl tab in the Output Window.
CWPERLI.MNU	Menu file that holds the menus for the right mouse click in Perl tab of Output Window.

The following files are found in the Hawk `PerlLib` subdirectory after installation:

Table 15-5. Files in the PerlLib Subdirectory

Filename	Purpose
CWP.DLL	Allows access to Hawk API's in the Perl script
CWP.PM	Exporter for Hawk API's
CARP.PM	Error routines
CONFIG.PM	Win32 Perl configuration
CWP.XS	Source Perl script used to rebuild <code>cwp .DLL</code>
DYNALoader.PM	Dynamically loads C libraries into Perl code

Table 15-5. Files in the PerlLib Subdirectory (Continued)

EXPORTER.PM	Default import method for modules
PERL5DB.PL	Perl 5.0 Debugger
TYPEMAP	Hawk variable types (for use with <code>CWP.XS</code>)
XSUBPPP.PL	Converts Perl XS code into C code (for use with <code>cwp.xs</code>)
TERM\CAP.PM	Perl termcap interface
TERM\COMPLETE.PM	Perl word completion module
TERM\READLINE.PM	Perl interface to various C<readline> packages.

Other Perl Resources

Much useful information can be derived from your nearest CPAN (Comprehensive Perl Archive Network) web site, or some of the Perl books by O'Reilly Press, namely ***Programming Perl***, and ***Learning Perl for Win32 Systems***. There are many mirror sites for CPAN, one of which is <ftp://ftp.cdrom.com/pub/perl/CPAN/>. Other useful URLs include <http://www.perl.org> for general Perl information and <http://www.activestate.com> for Perl for Win32 systems.

AppBasic

This section explains how to use the AppBasic Macro Language in Hawk to create your own macro functions.

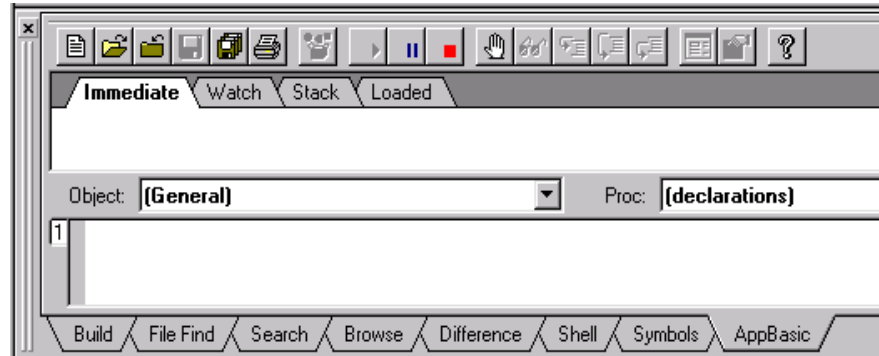
The Hawk AppBasic Macro Language is an interpreted language that is similar in definition and structure to Microsoft's Visual Basic. Through it, you have access to the Windows API, the Hawk API, and functions in independent DLLs.

Example files are supplied along with Hawk to aid you in getting started.

You will find them in Hawk's `MACROS` subdirectory. When you press on the `FileOpen` button on the `AppBasic Toolbar` for the first time, you will see a list of these sample `AppBasic Macros`. Several of these macros are covered in this section as well.

The advantage of an interpreted language is that no compiling is required. You can write your macros and use them immediately. By using the `AppBasic Window`, a tab on Hawk's `Output Window`, you can add break points and debug your macro, as well.

Figure 15-1. Output Window with AppBasic Tab Selected



Getting Started

To enable the AppBasic Macro Language, go to the **Customize -> Libraries** dialog, and you will see a list of optional packages or modules you can use with Hawk. Select the check box for AppBasic Macro Language. After loading the AppBasic package, you should be able to see an AppBasic tab on the Output window when it is visible. You can make the Output window visible by selecting it from the window menu.

Two Editors

You can select one of two editors to use to edit your AppBasic source code. You can use Hawk as your editor, or the Rich Text Editor.

If you use Hawk, you get the benefits of standard keystrokes and advanced features. If you use the Rich Text Editor, you get a very close emulation of the Visual Basic editor, including automatic case correction.

You can select which editor appears in the AppBasic window by enabling or disabling the **Rich Text Editor**. You will find this setting on the **Tools -> AppBasic Macros -> View** menu.

Special Keybindings

The following keybindings are in effect in the Rich Text Editor and cannot be changed. When using the Hawk Editor in AppBasic these keybindings are only valid when you have a .CWB file current. Otherwise, they revert to your original keybinding for your default keymap.

Table 15-6. Keybindings

Keystroke	Operation
C+ A	View Macro
C + E	View Immediate Tab Window

Table 15-6. Keybindings (Continued)

C + W	View Watch Tab Window
C + T	View Stack Tab Window
C + L	View Loaded Tab Window
%	Debug Run
S + %	Debug Stop
X	Debug Pause
*	Debug Step Into
S + *	Debug Step Over
C + *	Debug Step Out
&	Debug Step To
C+S+ (Debug Clear All Breakpoints
C + (Debug Add Watch
S +(Debug Quick Watch
(Debug Toggle Breakpoint
C + N	File New
C + O	File Open
C + -	File Close
C + S	File Save
C + P	File Print

Two Toolbars

AppBasic has two nearly identical toolbars that you can use. One toolbar, the one you see depicted in the preceding AppBasic Environment graphic, has a fixed location at the top of the AppBasic Window. We will refer to this toolbar as the AppBasic Window Toolbar.

The other toolbar is one of the Hawk Dockable Toolbars. You can position it anywhere on your screen, or dock it against any edge of the Hawk Client Area. We refer to this as the AppBasic Dockable Toolbar, to avoid confusion. You can enable this toolbar via the Toolbars dialog on the Customize menu.

Pop-up Menu

When you right-click in the AppBasic Window, a menu pops up that has a number of additional options. If you do not find what you are looking for on the Toolbar, or the Tools menu, refer to this menu.

Online Help

Online help is available for the AppBasic Window, the AppBasic Language, and its built-in functions:

- For help on using the AppBasic Window, or language syntax, press the Help button on the AppBasic Window Toolbar.
- For help on functions or subroutines, press the ! key anytime the cursor is in the AppBasic Window. Hawk will attempt to bring up help for the word at the cursor. If there is no word at the cursor, you can still browse a list of available functions and subroutines.

UserDialog Editor

AppBasic comes with its own dialog editor. A UserDialog is a dialog defined in a macro program. It is described within a Begin Dialog...End Dialog block. To create or edit a UserDialog graphically, place the cursor in a UserDialog block and press the Edit UserDialog button.

Object Browser

The Object Browser shows information about all the available special data types, particularly for OLE Automation. Select the label that you wish to look up, and click the Browse Object button.

If no label is selected, or the label is not found in the known libraries, the edit box at the top of the dialog will be empty. You may still browse the existing data types and methods.

Creating a Macro

To create an AppBasic macro, complete the following steps:

1. Select the AppBasic tab on the Output Window.
2. Right-click in the window and select Properties. In the Module Properties dialog, you can provide the names of the AppBasic functions you want to call from LibFunctionExec, the Hawk API Command, or bind to a Hawk button.

AppBasic refers to the functions as Handlers. A Handler exports the function for use by defining its return value, along with parameters and types. For those who are familiar with writing DLL extensions for Hawk, Handlers serve the same purpose as

calling `LibExport`. Once made available to Hawk in this way, Handlers can be assigned to keys or run from menus.

You can define and create Functions or Subs in the `Module Properties` dialog before or after creating a Handler. The handler must be added at some point, however, for the function to be called from Hawk.

1. When you have finished adding your functions, select **OK**. Then, on the `Modules Properties` dialog, select **Add**. The `Add Handler` dialog displays.

Creating a Handler

To create a Handler:

- In the `Add Handlers` dialog, name your function and declare its type.
- Functions and Subroutines must start with a letter and can then be followed by an underscore or letter.
- Type in any parameters you want.
- At this time, parameters cannot be added later, to do this you must delete the function and add it again.

Example:

From the `Add Handlers` dialog:

1. Enter `DoThis` as the name of the subroutine
2. Use a void return type and empty parameter list.
3. Press **OK** to return to the `Module Properties` dialog.
4. Press **OK** again.
5. When you are back in the `AppBasic` Tab, use the `Proc` list to select `DoThis`.

The `DoThis` subroutine will be inserted as follows:

```
Private Sub DoThis()
End Sub
```

6. Now add the following line inside the `DoThis` Sub.



```
MsgNotify "This is a test"
```

`MsgNotify` is a Hawk API function that notifies the user with a message.

The `DoThis` subroutine will now look as follows:

```
Private Sub DoThis()
    MsgNotify "This is a test"
End Sub
```

7. You should now save the macro to a file. By convention, the `.CWB` extension is used for `AppBasic` macros.
8. Now Run the macro. You may do this in any one of the following ways:

- Press the **Start/Resume** button on the AppBasic Window toolbar .
 - Press the **Run Current Macro** button on the AppBasic Dockable toolbar .
 - Press the right mouse click when over the AppBasic Output Window and select **Run**.
9. After running the macro, go to the Hawk menu **Tools** -> **API Command** and type: **DoThis**
 10. Click **OK**. A message box should appear with the words "This is a test" in it. .



It is not always expedient to load the macro by starting it in the **AppBasic Window**. When the macro is no longer in need of editing, it can be loaded using the **Load AppBasic Extension Macros** dialog, accessed by clicking **Load Macros** on the **AppBasic Macros** submenu of the **Tools** menu.

Object and Proc Dropdown Lists

The **Object** list shows all the objects for the current module. The object named **groups** together all of the procedures that are not part of any specific object.

The **Proc** list shows all the procedures for the current object. Selecting a procedure that is not bold inserts the proper procedure definition for that procedure.

Figure 15-2. The Object and Proc Dropdowns



Handlers you added in the **Properties** dialog will appear in the **Object** and **Proc** list.

Example

You create a function named `Srch_Backwards()`, then the **Object** list contains `Srch` and the **Proc** list contains `Backwards`.

The underscore divides the function name between the **Object** and **Proc** lists.

Private Sub Main

Any initialization can be done in a special subroutine named `Private Sub Main()`.

Example

```
Private Sub Main()
    Dim firstTime As Boolean
    Dim i As Integer

    firstTime = True
```



```
If (firstTime = True) Then
    ' no LibExports() needed here, use Modules Property
    ' Dialog And add handler as a replacement.
    firstTime = False
    Set SaveEvent = EventRegister(EVENT_SAVE_BUFFER,
    EVENT_NORMAL, "Buffer_Saved")

End If

End Sub
```

Tips on Creating Macros

When creating macros, consider the following tips:

- You may only edit macros that are not currently loaded for execution. To unload a macro, right-click on the `AppBasic` tab of the `Output Window` to bring up the `Show Loaded Modules` dialog. Removing the check from the check box in front of the filename will unload the macro. It can also be unloaded by calling `cwbUnloadFile(filename)` through the API Command prompt. If you try to edit a macro that is currently loaded an error message will appear, asking you to unload the module.
- When your cursor leaves a line of source code, it is automatically processed. You may note that capitalization has consequently changed, if you are using the Rich Text Editor.
- A dot in the left margin of the line indicates a break point. Break points may be toggled on/off, using the button on the `AppBasic` toolbar.

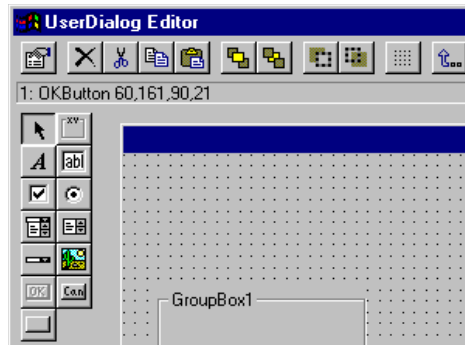
Creating a Modal User Dialog

To create a modal user dialog:

1. Place the cursor in the subroutine or function that you want to place the dialog.
2. Select one of the following:
 - `AppBasic` toolbar button `Edit UserDialog`
 - OR -
 - Hawk `AppBasic` toolbar button `Insert/Edit User Dialog`

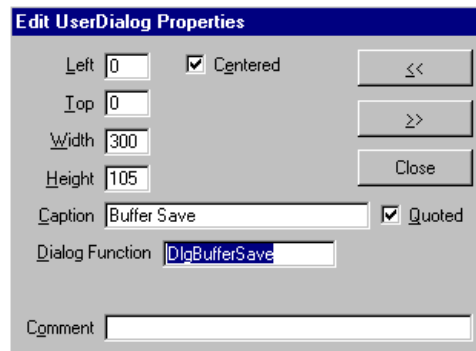
The `UserDialog Editor` displays:

Figure 15-3. UserDialog Editor



3. Select the **Edit Properties** toolbar button (or right-click the mouse button on the dialog that you are creating). The **Edit UserDialog Properties** screen displays.

Figure 15-4. Edit UserDialog Properties



4. Name your dialog in the **Caption** box.
5. Enter the name of the function that will be used to process the dialog in the **Dialog Function** box.
This is a very important step in the process. A question about creating the skeleton will not show up upon exiting the screen if you have not filled in the **Dialog Function** box.
6. Click the **Close** button to re-display the dialog in the **UserDialog Editor**.
7. Using the **OK** and **Can** tools on the vertical toolbar, add an **OK** and/or a **Cancel** button to the dialog.
A **Cancel** button is needed to activate the "x" system menu in the top right hand corner of the dialog.
8. Perform additional editing as desired.
9. Save and exit the **User Dialog Editor**, by clicking the **Save and Exit** button. A dialog will appear that asks: **Create the skeleton dialog function?**

10. Answer **YES** to this question. Your code should now look similar to the following:

```
Private Sub DoThis()
    MsgNotify "This is a test"
    Begin Dialog UserDialog 400,91,"DoThis Test dialog",.DoThisTest
    Text 30,28,330,28,"This is a test dialog.",.Text1
        OKButton 40,63,90,21
        CancelButton 160,63,90,21
    End Dialog
    Dim dlg As UserDialog
    Dialog dlg

End Sub

Rem See DialogFunc help topic for more information.
Private Function DoThisTest(DlgItem$, Action%, SuppValue%) As
Boolean
    Select Case Action%
        Case 1 ' Dialog box initialization
        Case 2 ' Value changing or button pressed
Rem DoThisTest = True ' Prevent button press from closing dialog
        Case 3 ' TextBox or ComboBox text changed
        Case 4 ' Focus changed
        Case 5 ' Idle
Rem DoThisTest = True ' Continue getting idle actions
    End Select
End Function
```

11. Change the following line of code from the following:

```
Dialog dlg
```

to the following:

```
bButtonPushed = Dialog(dlg)
```

See the topic *Dialog Instruction/Function* in the AppBasic online help for more information on this topic and the return values. You can access AppBasic help by pressing the "?" button on the AppBasic Window Toolbar, or by pressing ! anytime the cursor is in the AppBasic Window.



Modeless Dialogs are not available in AppBasic at this time.

The final sample follows:

```
Private Sub DoThis()
    MsgNotify "This is a test"
    Begin Dialog UserDialog 400,91,"DoThis Test dialog",.DoThisTest
    Text 30,28,330,28,"This is a test dialog.",.Text1
        OKButton 40,63,90,21
```

```

        CancelButton 160,63,90,21
    End Dialog
    Dim dlg As UserDialog
    bButtonPushed = Dialog(dlg)

End Sub

Rem See DialogFunc help topic for more information.
Private Function DoThisTest(DlgItem$, Action%, SuppValue%) As Boolean
    Select Case Action%
        Case 1 ' Dialog box initialization
        Case 2 ' Value changing or button pressed
    Rem DoThisTest = True ' Prevent button press from closing dialog
        Case 3 ' TextBox or ComboBox text changed
        Case 4 ' Focus changed
        Case 5 ' Idle
    Rem DoThisTest = True ' Continue getting idle actions
    End Select
End Function

```

Running the Macro

Run the macro to see if you have any syntax errors. After editing any syntax errors, go to the **Tools** -> **API Command** and execute **DoThis**. You will see a message box and then a dialog box show up.

The code within this subroutine will be run when the module is put into Run mode. Once a module is debugged, it can be loaded without being in AppBasic. The Hawk API command `cwbLoadFile "FileName"` will load the module and put it in run mode without being shown in AppBasic. You can bind the command `cwbLoadFile "FileName"` to a key, or add it to the [Editor] section of your CWRIGHT.INI configuration file, to load it during start up.



The macro can also be loaded by adding or enabling it in the Load Macros dialog, accessed by clicking **Load Macros** on the **AppBasic Macros** submenu of the **Tools** menu. Refer to the topic [Load Macros Dialog](#) for more information.

Creating an EventHandler in AppBasic

Events provide a method of interrupting program flow to allow a function or functions to have a timely effect. A number of events have been built into AppBasic using Hawk event handlers in order to add flexibility. One reason for this flexibility is that the number and names of the functions the event executes need not be known to the function that triggers the event.

The functions that are executed when a specified event occurs are called Event Handlers. Since most events originate with Hawk API functions, writing your own event

handler gives you access to the Hawk core. You can change the way critical functions work without rewriting Hawk.



Additional information on events can be found in the Hawk API (Functions) Help file under the section *Programming* and the topics *Events* and *Using Events*.

You can use events by following the three steps listed below:

1. Select the event that represents the action in which you wish to intervene.

An event handler list can be found in the Hawk API Help file under the topic *Using Events*. The sample event handler program ties itself to the event that occurs when a character is entered into a Hawk buffer.

2. Write an appropriate event handler function for that event.

The `EventHandler` definition must be "Global" or the event handler will become unregistered.

3. Register the function for execution at the event with `EventRegister`.

```
Set X=EventRegister(EVENT_CHAR_INSERTED, EVENT_NORMAL,
"EventTestHandler").
```



Deregistration of the event is handled automatically when the macro file is unloaded in Hawk. To stop an event handler before terminating the program (module), set returned `EventHandler Object` to "Nothing".



Additional information for event handlers can be found in the Hawk API Help file. Topics include *Event Handler* and *Events*.

Sample EHTEST.CWB

```
*****
' * EhTest.cwb
' *
' * Sample Event handler EVENT_CHAR_INSERTED, Message box ' ' ' *
pops up and displays the character that was pressed.
' *
' * Usage:
' * occurs automatically when the macro is loaded
' * because the event is registered in Private Sub Main().
*****
' Note!!!! EventHandler must be Global or event handler will
become unregistered.
Dim X As EventHandler

' This function needs to be executed for the handler to take '
affect. You can put the EventRegister in a main()
' subroutine if you want it to be available every time the
' macro is executed.
```

```
Private Sub Main()  
    Set X=EventRegister(EVENT_CHAR_INSERTED,EVENT_NORMAL,  
"EventTestHandler")  
End Sub  
Private Function EventTestHandler(ID As Long, Datap As Long) As  
Integer  
    MsgBox StringFromPointer(Datap)  
    EventTestHandler = 0  
End Function
```

Debugging Your AppBasic Macro

When you are ready to begin debugging an AppBasic macro, the following steps will help you get started:

1. Open the macro file for modification, if you have not already done so (Load it into the AppBasic Window).
2. Set a Break point perhaps at the first line of the Sub or Function you are debugging, by clicking the **Toggle Breakpoint** button (A dot should appear in the margin to the left of the line).
3. Enter Run mode by clicking the **Run, Step Over** or **Step Into** button. The Immediate, Watch, Stack and Loaded tabs then appears above the Edit Window. You can also press * to enter this mode if you are using the RTF editor, instead of the default Hawk edit window (Tools -> AppBasic Macros -> View -> Rich Text Editor).
4. Call the Sub or Function so that you reach the Breakpoint you have set. You can do this by selecting API Command from the Tools menu and entering the name of the Sub or Function.

Break Points

Toggle a break point on the current line.

- When you are debugging the macro and have hit a break point, notifications will be disabled in the other tabs in the **Output Window**. For example, if you have hit a break point in the macro and then you do a multiple search, the double clicking in the **Search Output Window** will be disabled until you have finished running the macro or stopped it.

Evaluate Expression and Add Watch

You may evaluate an expression, assign a value to a variable, or call a subroutine by typing commands in the `Immediate Window` when `AppBasic` is running a macro.

Table 15-7. Expressions

Command	Results (when you press <code>E</code>)
?<expr>	Shows the value of "expr"
<var> = <expr>	Changes the value of "var"
Set <var> = <expr>	Changes the reference of "var"
<subname> <args>	Calls a subroutine or built-in instruction
Trace	Toggles trace mode. Trace mode prints each statement in the <code>Immediate Window</code> when a macro/module is running.

The `Watch Window` displays the variables, functions and expressions that are calculated. Each time execution pauses, the value of each line in the `Watch Window` is updated.

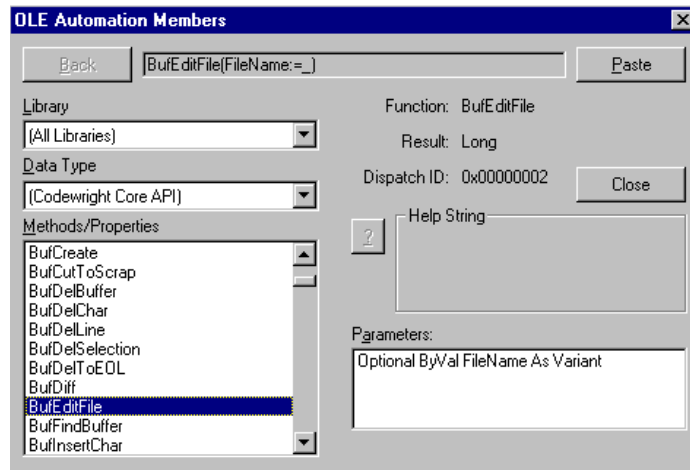
In addition, the following actions are available to you:

- The expression to the left of the "->" may be edited.
- Press `Enter` to update all of the values displayed, to reflect any changes you have made.
- Press `Ctrl-Enter` to delete the line.

Object Browser

The `Object Browser` shows information about all the special data types that are available.

Figure 15-5. OLE Automation Members Dialog



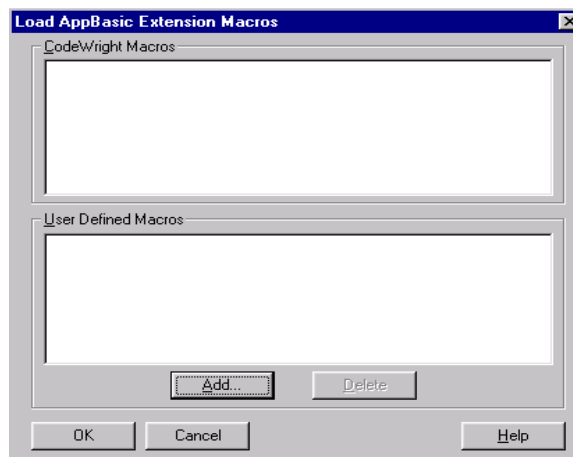
You can get to the Object Browser by pressing any of the following:

- The Browse Object button on the AppBasic Window toolbar.
- The Display Object Browser button on the AppBasic Dockable toolbar.
- The right mouse button, when over the AppBasic Output Window. Select **Object Browser** from the popup menu.

Load Macros Dialog

When you choose the **Tools -> AppBasic Macros -> Load Macros** menu item, you will be presented with the following dialog.

Figure 15-6. Load AppBasic Extension Macros Screen



This dialog allows you the opportunity to load the sample macros or user defined macros into memory.

AppBasic Samples Supplied

A list of the `AppBasic` modules supplied with Hawk follows, along with usage notes. Most of these samples were not intended to perform a useful task, but rather to show how features of Hawk are accessed through `AppBasic`.

- `Baskeywd.cwb`

The function `AddBasKeywords` adds the keyword `Integer` to BASIC language `ChromaCoding` (syntax coloring).

Usage:

```
AddBasKeywords ()
```

- `Colsum.cwb`

`SumSelectedText` adds the selected column of numbers and outputs the total on Hawk's status line.

Usage:

```
SumSelectedText (1)
```

- `Diffdir.cwb`

`DiffDirectory` differences the file in the reference directory against the files in the target directory. This uses a `Modal` dialog to display the files being differenced.

Usage:

```
DiffDirectory ("c:\testdir1", "c:\testdir2")
```

- `Dispname.cwb`

`cwbDisplayFileName` displays the name of the current file on the status line. An asterisk " * " follows the filename when the file contains unsaved edits.

Usage:

```
cwbDisplayFileName ()
```

- `EditNext.cwb`

`cwbEditNextBuffer` makes the next buffer in the list of buffers current.

Usage:

```
cwbEditNextBuffer ()
```

- `EditPrev.cwb`

Edit the previous buffer in the list of buffers.

Usage:

```
cwbEditPrevBuffer ()
```

- `EhTest.cwb`

When this macro is loaded, a message box pops up every time a key is pressed, displaying the character that was pressed. The function is stored in the macro's main section; therefore, it launches automatically when the macro is loaded.

Usage:

```
Main ()
```

- GotoLine.cwb
cwbGotoLine prompts on the status bar for the number of a line to go to.
Usage:

```
cwbGotoLine()
```
- HashTest.cwb
A hash table is created automatically by the main function when the macro is loaded.
Usage:

```
Main()
```
- SaveHdl.cwb
When loaded, this module adds an `EVENT_SAVE_BUFFER` event handler, which pops up a dialog whenever a buffer is saved. This program used the `Main()` subroutine to make the handler available when this program is running. No need to call a function to start it up. The `Main()` subroutine is the first function to be executed when any macro language file is loaded.
- SnapVirt.cwb
Snap up to first character on end of line skipping virtual space.
Usage:

```
cwbSnapVirtual()
```
- SrchRepl.cwb
Uses `SearchTranslate()` to execute multiple search and replaces on a buffer, to test this macro use the "From:" text below and put it in a buffer for the replacement.
Translate the following text,
From:

```
West Siberian plains.<252>The European part Is covered
<SUBHEAD>Soviet Period</SUBHEAD>
/par /Tab Union of Soviet Socialist
Republics<252><252><252>
```


To:

```
West Siberian plains.
/par /Tab The European part Is covered <SUBHEAD>Soviet
Period</SUBHEAD>
/par /Tab Union of Soviet Socialist Republics
```


Usage:

```
Search_Replace_Buffer()
```
- Stripwt.cwb
Strips trailing white space (" " or \t) from the end of a range of lines in a buffer.
Usage:

```
cwbStripTrailingWhite(1, 20, 5)
```

- Test.cwb

This module contains several functions to test a return value's type. Use this on commands entered at the `Tools -> API Command` prompt, preceded by a `?`.

Example

Using the command `?TestBool` give you the following:

```
Return:-1 (hex ffffffff), type:int
Usage (in Tools -> API Command dialog):
```

```
?TestBool()
?TestInt()
?TestLong()
?TestString()
```

- Wordcnt.cwb

`CountPhrase` counts the number of occurrences of a specified string that are found in the current document. A `Dialog` prints out the results (Uses `Debug.Print`).

Usage:

```
CountPhrase("the")
```

- ZoomWin.cwb

`cwbZoomWindow` toggles the window between `Maximized` and `Restored`. Uses `SendMessage()` windows call.

Usage:

```
cwbZoomWindow()
```

AppBasic-related API Commands

The following useful API commands are available from the Hawk API Command prompt (`Tools -> API Command`):

- `cwbLoadFile <moduleName>`

Once an AppBasic module contains no syntax errors, it can be loaded for execution. It can then be run without the use of the AppBasic window.

- `cwbLoadFile "FileName"`

Will load the module and make its exported functions (handlers) available for running. You can also bind keys and buttons to the handlers in the loaded Macro file.

- `cwbUnloadFile (<moduleName>)`

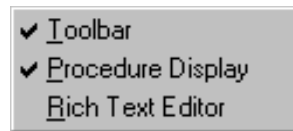
An AppBasic module cannot be modified while it is loaded for execution. If you have loaded the file using `cwbLoadFile()` from your `CWRIGHT.INI` file or from a button and you want to now edit the file in AppBasic you will need to unload the file using `cwbUnloadFile()`.

AppBasic Window Configuration

Most users will find the standard configuration of the AppBasic Window satisfactory. If desired, however, you can turn off the AppBasic Window Toolbar and the AppBasic Object and Proc drop down lists that appear at the top of the AppBasic Window.

To change the appearance of the AppBasic Window, go to the **Tools** -> **AppBasic Macros** -> **View** menu. Here you can enable or disable the AppBasic Window Toolbar, the AppBasic Object and Proc drop down list, or change from a Hawk Edit Window to the WinWrap Rich Text Editor.

Figure 15-7. Menu



This configuration can also be set by right-clicking over the AppBasic Window, and then selecting **View** from the pop-up menu.

If you wish to change the configuration of the window programmatically, the following Hawk API commands will assist you. These commands may be issued by the API Command dialog (Command Key) on the Tools menu. If you find them useful, you may make your settings more permanent by modifying similar commands in the MWHAWK.INI configuration file.

- `cwbShowToolbar (int nShow)`

This function will Hide/Show the Standard AppBasic Toolbar.

Example

```
cwbShowToolbar(0)
cwbShowToolbar(1)
```

The first example will cause the toolbar to be hidden. The second example will show the toolbar. Any positive integer value for the parameter will display the toolbar.

- `cwbShowProcDisplay (int nShow)`

This function will Hide/Show the Object and Proc dropdown lists.

Example

```
cwbShowProcDisplay(0)
cwbShowProcDisplay(1)
```

The first example will cause the Object and Proc dropdown lists to be hidden. The second example displays them. Any positive integer value for the parameter will display the Object and Proc dropdown lists.

- `cwbCreateCWWindow(int bCWWindow)`

This function is used to change the Editor in AppBasic. There are two choices: one is to make it a Hawk Editor and the other is a Rich Text Editor.

Example

To change to a Hawk Editor call the following:

```
cwbCreateCWWindow( 1 ) or cwbCreateCWWindow( TRUE )
```

Example

To change to Rich Text Editor call the following:

```
cwbCreateCWWindow( 0 ) or cwbCreateCWWindow( FALSE )
```

Example Configuration File Settings

The AppBasic window's configuration is stored in the MWHAWK.INI file, in the [AppBasic Setup] section as shown below:

```
[AppBasic Setup]
cwbShowProcDisplay=2
cwbShowToolbar=1
```

Exported Functions in CWBASIC.DLL

Refer to the following functions:

- cwbAddHandler(LPSTR lpszHandlerDef, LPSTR lpszModule)

Adds a callable Sub/Function to AppBasic module – equivalent to LibExport

```
lpdzHandlerDef - Sub/Function Prototype ex: Private Sub
CallMe(FName As String)
lpszmodule - cwb file containing this Sub/Function (Full
Path) ex: c:\cw32\colsum.cwb.
```

- cwbToggleBreakPoint(void)

Toggles a break point in currently edited module at the current line.

- cwbEditFile(LPSTR lpszFile)

Loads a .CWB file into the AppBasic editor.

- cwbLoadFile(LPSTR lpszFile)

Loads and runs AppBasic file.

- cwbUnloadFile (LPSTR lpszFile)

Unloads File from Engine (Stop Run)

- cwbExecuteCommand(long cmd)

Executes an AppBasic Operational Command.

Table 15-8. Commands

cmdFileNew = 0	CmdFileOpen = 1	cmdFileSave = 2
cmdFileSaveAs = 3	CmdFilePrint = 4	cmdFilePrintSetup = 5
cmdMacroRun = 6	CmdMacroPause = 7	cmdMacroEnd = 8
cmdDebugStepInto = 9	CmdDebugStepOver = 10	cmdDebugStepTo = 11
cmdDebugBreak = 12	CmdDebugQuickWatch = 13	cmdDebugAddWatch = 14
cmdDebugBrowse = 15	CmdDebugSetNext = 16	cmdDebugShowNext = 17
cmdHelpApp = 18	CmdHelpLanguage = 19	cmdHelpTopic = 20
cmdHelpAbout = 21	CmdEditUndo = 22	cmdEditCut = 23
cmdEditCopy = 24	CmdEditPaste = 25	cmdEditFind = 26
cmdEditReplace = 27	CmdEditAgain = 28	cmdEditFont = 29
cmdEditDelete = 30	CmdEditSelectAll = 31	cmdEditUserDialog = 32
cmdFileClose = 33	CmdFileSaveAll = 34	cmdDebugStepOut = 35
cmdSheetOpenUses = 36	CmdSheetCloseAll = 37	cmdSheet1 = 38
cmdSheet2 = 39	CmdSheet3 = 40	cmdSheet4 = 41
cmdSheet5 = 42	CmdSheet6 = 43	cmdSheet7 = 44
cmdSheet8 = 45	CmdSheet9 = 46	cmdEditProperties = 50
CmdFileNewObjectModule = 48	CmdFileNewClassModule = 49	cmdFileNewCodeModule = 47

- cwShowToolBar (int nShow)
Hide/Show Standard AppBasic Toolbar.
- cwShowProcDisplay (int nShow)

Hide/Show Object and Proc Dropdown Lists.

API (C-like) Macros

Hawk API commands are function calls that can be made interactively. This means that they can be assigned to keys, menu items, buttons, popup context menus and issued from the API command line. Hawk API Macros build upon this capability to provide a quick and simple way to write extensions for Hawk.

If you have previously used, or read about using API commands interactively, the following differences should be noted if they are being used from API macros:

- You can nest API commands as parameters to other API Commands in macros.
- All functions must be followed by opening and closing parentheses around the parameter list, if any.
- Parameters must be separated by commas.

API Macros Defined

API Macros are one or more sequential, iterative, or conditional statements that are given a name. These statements often contain Hawk API function calls. They are stored in a file, `MWHAWK.MAC`, and may be recalled by name for execution within Hawk.

A ChromaCoding lexer called `API Macro` has been developed to make working with the language easier. Without any special changes you should be able to do the following:

- Load the `MWHAWK.MAC` file into Hawk.
- See it correctly ChromaCoded.
- Compile it with the menu or button

Parse errors from the `Build` tab of the `Output Window`.

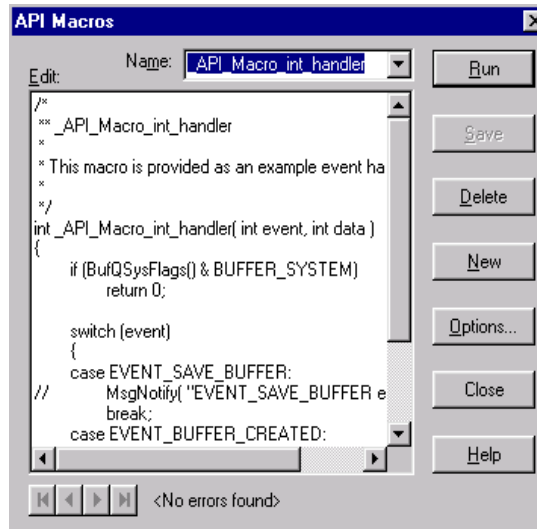


For more information on Lexers, refer to the topic [ChromaCoding Lexers](#) in [Chapter 5 View Setups and Language Support](#).

Getting Started with API Macros

The `Tools -> API Macros` dialog lets you create, edit and test API Macros.

Figure 15-8. API Macros Dialog



Creating a Macro

To create an API macro, complete the following steps:

1. Name the macro in the **Name** edit box.
2. Type in the API commands you wish to execute into the **Edit** box.
3. Press the **Save** button. Pressing the **Run** button will send away the dialog. Any unsaved edits are automatically saved by default.

Editing a Macro

To edit a macro previously created, select its name from the **Name** listbox. The text of the macro will appear in the **Edit** box and you can proceed to make your changes. Pressing either **Run** or **Save** will save your changes to disk.

Each time a macro is saved it is checked for errors. If there are errors, use the four buttons at the lower left of the dialog to parse the position and text for each error.

Special Editing Keystrokes

In addition to the buttons, the next error can be parsed by using **SCY**. To insert a **<Tab>** character, use **CT**. As in a Hawk window, press **!** to bring up help for the function under the cursor.

Editing API Macros in a Hawk Window

You can edit and create API Macros in a standard Hawk edit window. This gives you access to Hawk's advanced editing capabilities, error parsing and ChromaCoding.

Complete the following steps:

1. Open the file **MWHAWK.MAC** into Hawk.
2. Create a new macro using the following syntax:


```
[API Macro:<macro_name>]
```

- OR -

Find the location of the macro to edit by searching for `[API Macro:<macro_name>]`. The macro is contained in the lines following its name `<macro_name>`, but before the next bracketed section, if any.

3. Make any necessary edits.
4. Save the file before running the macro, since macros are always read from the file.

Setup Editing in a Hawk Window

Complete the following:

1. Use the API Macro ChromaCoding lexer in `Customize -> Language -> Coloring` to get ChromaCoding for API Macros that are edited in Hawk edit windows. Associate the lexer with a `.mac` file type, which should be added to the file type list on the left-hand side of the `Edit` dialog.
2. To set up the compiler for API macros: Enter `!MacroMakeErrorFile` in the `Compiler Name` field for the `Compiler` category of tools in `Project -> Settings -> Tools`.
3. In the `Project -> Settings -> Errors` dialog, choose `_APIMacroInfo` for the error parser.

Running a Macro

The most common way to run a macro is from the menu or dialog. Macros can also be executed from within Hawk in as many ways as API functions can be. Just use the macro's name, and optionally its parameters, as if it were a Hawk API function.

Language Definition

For further information on the API Macro language, refer to the following sections:

- [Comments](#)
- [Variables](#)
- [Literal Values](#)
- [Expressions](#)
- [Statements and Statement Blocks](#)
- [Program Flow of Control Structures](#)

Comments

Comments can be used almost anywhere, including within statements.

- C style multi-line comments may be used in API Macros. The comment starts with the forward slash and asterisk, `'/*'`, and ends with the asterisk and forward slash, `'*/'`. This kind of comment can cover a partial line, or many lines. Nested comments are not supported.

- C++ style comments may be used in API Macros. When two forward slashes, '//', appear on a line, the remainder of the line is ignored.

Identifier Naming Rules

Identifiers include macro and variable names. The first character must be an alphabetical or underscore (`_`) character. The remaining characters can be alphanumeric or the underscore character.

Variables

There are two types of variables available, 'int' and 'string':

- 'int' type variables are long signed integer numbers. You can use the 'int' type for accessing most of the parameter and return types of Hawk's API commands. These include `char`, `int`, `long`, `UINT`, `WORD`, `DWORD`, `BOOL`, `HBUFFER`, `HWND`, etc.
- 'string' type variables are NULL terminated character arrays. They are automatically allocated and freed as needed. Use the 'string' type when accessing parameter and return types of `LPSTR` and `LPMSTR`. Strings must start and end with the quote character (`"`).

Variables must be declared before they can be used. They can be assigned a value in the same statement they are declared in. Here are some example declarations;

```
int i;  
int j = 1;  
string s;  
string a = "abc";
```

Two of these examples include initialization. Macro variables are never uninitialized. If they do not have a value assigned to them, their value will be zero or NULL.

static

Use the 'static' keyword directly before the 'int' or 'string' type keyword. Static variables maintain their values between macro calls.

If you wish the initialized value of the variable to be non-zero, be sure to follow it with an assignment expression. This expression cannot contain any macro or function calls.

global

Use the 'global' keyword directly before the 'int' or 'string' type keyword. Like static variables, global variables maintain their values between macro calls. They have the additional benefit of being available in more than one macro.

Each macro that uses a global type variable must declare it. For each declaration, use the same assignment value for initialization. Otherwise, the starting value may be undetermined, since it would depend on which macro is compiled (used) first.

Parameters and Return Type

Declare parameter variables at the beginning of a macro. Use the following format for the macro declaration:

```
<ret_type> <macro_name>(<type> <param1_name>,  
<type><param2_name>, ... )  
{  
    <statement>;  
    // more lines here  
    return <expression>;  
}
```

The following formatting guidelines apply when working with API Macros:

- For backwards compatibility the parameters line and the outside braces enclosing the macro are optional.
- The `<ret_type>` part of the parameters line is required. If the macro returns no value, use the special type `'void'`. Otherwise use either `'int'` or `'string'` here.
- The `<macro_name>` part is optional since it serves no functional purpose. However to maintain good C style form, use the same name here as the macro name shown in the dialog's combo box.
- The left and right parentheses frequently contain one or more comma-separated formal parameters. Each formal parameter variable begins with either `'int'` or `'string'`, and ends with the identifier naming it. If no parameters are used, use the special `'void'` type alone between the left and right parentheses.
- If the return type is not `'void'`:
 - Use the return statement at the end of the macro.
 - Include an expression in the return statement to use for the return value.

Literal Values

Literal values can include numbers, characters and strings.

Number Literal Formats

Decimal, hexadecimal and octal number formats are supported. Hexadecimal numbers must start with `'0x'`. Octal numbers must start with `'0'`. All other numbers are considered decimal.

Character Literals

Character literals are just an alternate method of specifying a literal number. Delimit character literals at the beginning and end with a single quote (`'`) character.

Character literals usually contain one character but will have two or three if the first one is the backslash (`\`) escape character. Use the backslash before a contained single quote if you wish the single quote to represent the character literal value.

String Literals

String literals are sequences of characters that are delimited at the beginning and end by the double quote (") character. To include the quote in the string, precede it with the backslash (\) escape character. String literals can continue onto following lines if the last character in the line is a backslash.

Escape Sequences

The following are escape sequences for both `string` and `char` literals:

Table 15-9. Escape Sequences

Literal	Escape Sequence
<code>\n</code>	New line (<CR><LF> or <LF>, depending on how it is defined for the document)
<code>\t</code>	Horizontal tab
<code>\b</code>	Backspace, Ctrl-H
<code>\r</code>	Carriage return
<code>\f</code>	Form feed, 0x0c
<code>\nnn</code>	Octal value between 0 and 0377
<code>\xnn</code>	Hexadecimal digit between 0x00 and 0xFF
<code>\m</code>	The literal character m.

- If you want to use the backslash in function calls that expect a single backslash in the string, be sure to double it in the literal string.

Automatic Type Conversion

Values are automatically converted to another type when used where that type is expected.

Examples

- If a `'string'` type is used where an `'int'` is expected, this is how it is converted:

If the string begins with decimal, octal or hexadecimal characters, they are converted for the number. If not, the value will be one if the string contains at least one character and zero if it is empty.
- If the `'int'` type is used where a `'string'` is expected, the value is converted to decimal characters.

Expressions

Expressions are made up of operators and operands. Operators can be variables, literals, operators, and function or macro calls; support is provided for all C operators. Numeric, logical and string expressions are allowed. Expressions can be used as function call parameters.

Assignments and Operators

Assignments take the form:

```
<variable> <assign_op> <expression>
```

Assignments are allowed to begin at three places in a statement: at the beginning of the statement, directly following a left parenthesis or directly following a comma. Multiple assignments are allowed in the same expression.

Example

```
a = b = c = 0;
```

The assignment operators available include the following:

Table 15-10. Assignment Operators

Operator	Description
=	Simple assignment
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulo and assign
<<=	Shift bits left and assign
>>=	Shift bits right and assign

The numerical operators available include the following:

Table 15-11. Numerical Operators

Operator	Description
+	Addition

Table 15-11. Numerical Operators (Continued)

-	Subtraction
*	Multiplication
/	Division
%	Modulo (remainder)
<<	Shift bits left
>>	Shift bits right
&	Bitwise AND
	Bitwise OR
^	Bitwise Exclusive OR
~	Bitwise Complement

Logical expressions evaluate to a one or zero, TRUE or FALSE, respectively. Logical operators include the following:

Table 15-12. Logical Operators

Operator	Description
=	Equivalence
!=	Non-equivalence
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
&&	AND
	OR
!	Negation

String Operator Rules

Some operators have special meaning when their operands are strings. Use '+' to Concatenate; use '+=' to concatenate and assign.

String concatenation is the joining of two strings where the string following the operator is appended to the string before the operator. If the left operand is of type 'string', string concatenation is done, otherwise numerical addition is done. The right operand is converted as needed for either form. This capability makes it easy to compose output strings containing the descriptions and values of variables.

The following are logical comparison operations:

- == TRUE if string1 and string2 match
- != TRUE if string1 and string2 don't match
- > TRUE if string1 is alphabetically after string2
- < TRUE if string1 is alphabetically before string2
- >= TRUE if string1 is alphabetically after string2 or matches
- <= TRUE if string1 is alphabetically before string2 or matches

String comparison operations require that both operands be strings. Otherwise the numerical comparison is done. The single string operand is converted to type 'int' if needed.

By default, string comparisons are case insensitive (ignore case is on).

- To change case sensitivity for the current invocation of a macro, put the call below in your macro:

```
MacroIgcase (FALSE) ;
```

- To query the current state use the following:

```
MacroIgcase (-1)
```

Other Operators

Refer to the following descriptions of other available operators:

Table 15-13. Other API Macro Operators

Operator	Description
,	<p>Comma operator. This operator is used for two purposes:</p> <ul style="list-style-type: none"> • Separates parameters in the calls to other macros or functions. (most commonly). • Separates sub-expressions within a larger expression. This use is often seen in the 'for' statement, where it allows multiple sub-expressions in any of the three semicolon-separated sections that control the 'for' statement. <p>The value taken for a sequence of comma-delimited sub-expressions is that of the right-most sub-expression.</p>

Table 15-13. Other API Macro Operators (Continued)

?:	<p>Conditional operator. This operator takes the form:</p> <pre><test_expr> ? <true_expr> : <false_expr></pre> <p>If <test_expr> evaluates as non-zero, the <true_expr> is executed and its value is taken. Otherwise, the <false_expr> is executed and its value is taken. Although this operator can be used without restriction within expressions, it is commonly used as the right part of assignment statements.</p> <pre>bValue = bTest ? bVal1 : bVal2;</pre>
++	<p>Pre-increment or post-increment operator. This operator can only be used directly preceding or trailing an 'int' type variable:</p> <ul style="list-style-type: none"> • When used before the variable, the variable's value is increased by one before it is used within an expression, if any. • When used after the variable, the variable's value is taken first for use in an expression, then the variable's value has one added to it.
--	<p>Pre-decrement or post-decrement operator. This operator can only be used directly preceding or trailing an 'int' type variable:</p> <ul style="list-style-type: none"> • When used before the variable, the variable's value is decreased by one before it is used within an expression, if any. • When used after the variable, the variable's value is taken first for use in an expression, then the variable's value has one subtracted from it.

Operator Precedence Rules

Operators with the highest precedence are evaluated first. These operators are binary (requiring left and right operands) unless otherwise indicated.

Associativity indicates the order of evaluation when two operators have the same precedence. For example, most operators evaluate starting at the left in an expression, processing right.

Table 15-14. Operator Associativity

Operators (from high to low precedence)	Associativity
()	none
- ~ ! ++ -- (all unary)	R to L
* / %	L to R
<< >>	L to R
+ -	L to R

Table 15-14. Operator Associativity (Continued)

< > <= >=	L to R
== !=	L to R
&	L to R
^	L to R
	L to R
&&	L to R
	L to R
? : (trinary operator)	L to R
= *= /= %= += -= <<= >>= &= = ^=	R to L
, (comma operator)	L to R

Parentheses in Expressions

Left and right parentheses, '(' and ')', may be used to group expression parts to set evaluation preference.

Short Circuiting

The '&&' and the '||' operators have special evaluation rules. The idea of short circuiting is that often only the left operand needs to be executed and evaluated.

- For the '&&' operator, if the left operand evaluates as zero, the value of the operation must be zero (`FALSE`) so the right operand can be ignored.
- For the '||' operator, if the left operand evaluates as non-zero, the value of the operation must be one (`TRUE`), so the right operand can be ignored.

Function Calls

Function calls, including those to Hawk API functions and macros, take the following form:

```
<macro_name> <parameter1>, <parameter2>, ... <parameterN>
```

The following rules apply:

- Spaces and tabs are allowed after the macro name and between parameters.
- Parameters must be separated by commas.

- You may supply fewer parameters than a function is defined to use. The remaining parameters will be supplied for you as zeros or NULLs.



- See the discussion on [Expressions](#), in this chapter, for what can be used as parameters. Macro recursion is supported.

Exiting and Macro Return Values

Use the return statement to exit a macro before the last line and if the macro has a return type, return a value. End the return statement with a semicolon.

For backward compatibility, if there is no specified macro return type, the return value is optional. For what is allowed, see the discussion on [Expressions](#) in this chapter.

If the expression does not evaluate to the type expected by the macro's defined return type, the value is automatically converted. See the preceding discussion on [Automatic Type Conversion](#), in this chapter.

Statements and Statement Blocks

Statements are expressions including optional preceding assignments, ended by a semicolon. Statements can span multiple lines if needed. More than one statement can share a line, but it is better form to have them on separate lines. A statement need not contain anything more than a semicolon.

Statement blocks are a series of statements. They are normally used in conjunction with control structures. Use curly braces '{ ' and ' }', to begin and end statement blocks, respectively. A statement block can be used anywhere a single statement can be used.

Program Flow of Control Structures

The available program flow control structures are of conditional and iterative types. Both conditional and iterative statements can be used anywhere statements might be used.

Conditional Statements

Examples of conditional statements include:

- 'if' statement

```
if (<expression>
    <statement>;
```

- 'if-else' statement

```
if (<expression>
    <statement>;
else
    <statement>;
```

Testing for several conditions can be done by using the 'if-else' statement in place of the statement following the 'else'. The following does this and also uses a statement block:

- 'if' (<expression one>)

```
{
    <statement>;
    <statement>;
}
else if (<expression two>)
    <statement>;
else if (<expression three>)
    <statement>;
else
    <statement>;
```

- 'switch' statement

```
switch (<test_expression>)
{
    case <constant_expression1>:
        <statement>;
        <statement>;
        break;
    case <constant_expression2>:
        break;
default:
    <statement>;
    break;
}
```

The 'switch' statement executes specific statements based on the value of a test expression. Specify which statements to execute for given values via the 'case' lines. There can be many 'case' lines, but each must have a unique value from the constant expression.

These constant expressions must be able to be evaluated at compile time. They must not contain function or macro calls. They must not contain variables or string literals.

The 'break' statement line following each 'case' section is optional. Without a 'break', execution continues into the following 'case' or 'default' section. The optional 'default' section is executed if none of the cases are matched by the test expression. The braces are required.

Iterative Statements

Refer to the following types of iterative statements:

- 'while' statement

```
while (<expression>)
    <statement>;
```

The 'while' statement allows you to execute a statement repeatedly.

The <expression> is evaluated before each iteration:

- If non-zero, the contained statement is executed.
- If zero, execution proceeds with the statement following the 'while' statement.

- 'do-while' statement

```
do
{
    <statement>;
} while (<expression>;
```

The 'do-while' statement also allows you to execute a statement repeatedly. The <expression> is evaluated after each iteration. The 'do-while' statement will execute its contained statements at least once. The braces are required.

- 'for' statement

```
for (<init_expression>; <test_expression>;
    incr_expression)
<statement>;
```

The 'for' statement contains three parts within the parenthesis:

- The first part, <init_expression>, only executes once, but before any other part of the for statement.
- The second part, <test_expression>, is executed before each iteration of the loop. If its value is non-zero, the contained <statement> is executed. If <test_expression> is missing, the contained <statement> always executes.
- The third part, the <incr_expression>, is executed after each loop iteration.

Each of the three parts is optional. If you leave out the <test expression>, it is common to end the loop from within the contained statement section using the 'break' statement.

- 'break' statement

```
break;
```

The 'break' statement causes execution to resume immediately following the innermost statement containing 'switch', 'while', 'do-while' or 'for'.

- 'continue' statement

```
continue;
```

The 'continue' statement causes execution to not finish the current iteration of the innermost containing loop; instead, the next iteration is begun. In the 'while' and 'do-while' statements, the next evaluated is the <test expression>. In the 'for' statement, the <increment expression> is evaluated next.

Run-time Error Handling

API macros protect against several kinds of run-time errors:

- ctrl-break stop

If you have written a locked loop in your macro, use the C [Break] keystroke to terminate it.

- Divide by zero error

If the macro attempts a divide or modulo by zero, you will get this error.

- Runaway recursion protection

Recursion macro code can sometimes have the problem of not ending the recursion correctly. This can lead to an ever-expanding stack. Each time an API macro is called from another, the level of call is checked to see if it is greater than the allowed maximum (default is 50). This maximum level can be adjusted with the function `MacroMaxRecursion`.

Hawk Event Handling

Three special macros have been added to `MWHAWK.MAC` to facilitate event handling by API macros:

- `_Init_API_Macro` – This macro is automatically called when Hawk starts up. To register events you want to handle, put calls to `EventRegister` here. You can also put other code here you want run at startup.
- `_API_Macro_int_handler` – This macro is an example handler for when the event data is known not to be type 'LPSTR'. Use this macro for most events.
- `_API_Macro_string_handler` – This macro is an example handler for when the event data is known to be the type 'LPSTR'.

On events where the data is a pointer of type other than 'LPSTR', you can still trap the event but will not be able to access the data.

Differences between API Macros and C

The following are included in C, but not in API macros:

- types 'char', 'short', 'float', 'double', 'unsigned'
- arrays and pointers
- Typedefs, enums, structs and unions
- variable scope limited to statement blocks
- preprocessor capability
- "... " parameter type (`printf`)
- 'goto' statement and labels
- link phase and libraries

The following are included in API macros, but not in C:

- variables needn't be declared at function top (like C++)
- lesser number of parameters in calls allowed (like C++)
- string type and special operator capability

- automatic memory management (for 'string' type)
- compile on demand (first time run)
- run-time error handling
- global variables are declared within macros

String Functions

Since library functions are not available in API Macros, a number of string functions are made available through the Hawk API for building and comparing strings.

- `LPMSTR StringApnd(LPSTR str1, LPSTR str2);`
Append `str2` onto `str1`. Neither string is freed.
- `LPMSTR StringNApnd(LPSTR str1, LPSTR str2, int len2);`
Append `len2` bytes of `str2` onto `str1`. Neither string is freed.
- `int StringLength(LPSTR str);`
Return the length of `str`.
- `int StringCompare(LPSTR str1, LPSTR str2);`
Returns `< 0` if `str1` is alphabetically before `str2`. Returns `> 0` if `str1` is alphabetically after `str2`. Returns `0` if `str1` matches `str2`.
- `int StringICompare(LPSTR str1, LPSTR str2);`
Like `StringCompare` but ignores case.
- `int StringNCompare(LPSTR str1, LPSTR str2, int len2);`
Like `StringCompare` but only compares `len2` bytes.
- `int StringNICompare(LPSTR str1, LPSTR str2, int len2);`
Like `StringCompare` but ignores case and only compares `len2` bytes.
- `LPMSTR StrAscii(int ch);`
Converts a character into a single character length string.
- `LPMSTR StrItoA(long value, int radix);`
Converts a number into a string. Use a radix of 10 for decimal string.
- `LPMSTR StrLtrim(LPSTR string, LPSTR cset);`
Trims characters in `cset` off the left of string.
- `LPMSTR StrTrim(LPSTR string, LPSTR cset);`
Trims characters in `cset` off the right of string.
- `LPMSTR StrSubStr(LPSTR string, int start, int end);`
Return a substring out of string.
- `LPMSTR StrFormatDate(long t, LPSTR fmtStr);`
Formats a time/date value into a string.
- `LPMSTR TransformFilename(LPSTR filename, LPSTR spec);`
Documented in the online help.

- `BOOL StrFileMatch(LPSTR fpattern, LPSTR fname, BOOL igcase);`
Determine if a filename is matched by a pattern.

Making DLL Add-Ons

Loading DLLs, or Add-Ons, interactively from the `Customize -> Libraries` dialog will add functionality to Hawk. The functionality that is added depends on what the DLL in question is designed to do. Many Add-Ons are available in a list of `Hawk Libraries` in the `Libraries` dialog. Those Add-Ons are quickly loaded by placing check marks next to the desired Add-On. More Add-Ons are available on the Hawk CD.

Consider adding a function by modifying a Hawk DLL or creating a new DLL, only if all of the following are true:

- Hawk can not perform a particular function.
- There is no existing Add-On or macro that will add that function.
- Creating a macro is out of the question.

Source code for most Hawk DLLs is provided with a full installation of Hawk. A full Hawk installation also installs a `sample` subdirectory in the Hawk directory that contains source code for an essentially empty DLL, which can easily be modified to add any function desired.

This section has three major provisions for changing or adding capabilities to Hawk through DLLs:

- It provides direction to the source code to be changed or added to.
- It provides some assistance for making the changes or additions.
- It provides tips on how to compile and make use of the changes or additions that have been made.

The first step to changing Hawk is to understand its anatomy.

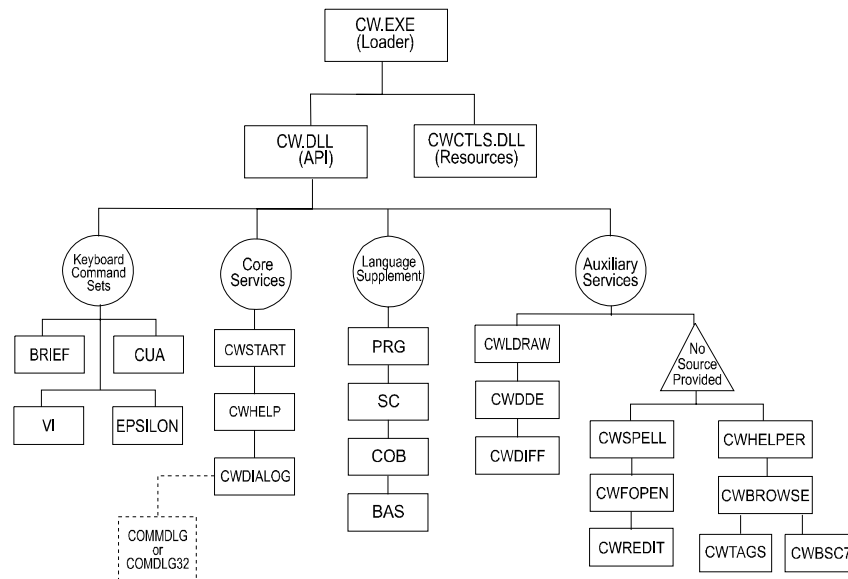
The primary executable, `MWHAWK.EXE`, contains the Hawk Application Programming Interface (API). These are the functions and capabilities that are basic to Hawk's operation. Although you cannot rewrite the functions in the API, you can replace them, in a limited fashion. However, please note that the source code for main executable is not supplied.

External to the primary executable are several Dynamically Linked Libraries (DLLs) which provide services in several areas, including the following:

- core services
- supplemental language support
- keyboard command sets
- auxiliary services

The following illustration depicts the organization of the Hawk software system. The auxiliary and language supplements are not complete. Hawk offers more in these areas than would fit on a page. However, the picture gives an accurate idea of the Hawk architecture.

Figure 15-9. Hawk Architecture



Core Services

The core services are those that Hawk always expects to have available. Some of these services are so essential that Hawk is unable to properly initialize without them. For example, you could use Hawk if no keyboard command sets were available — you could even build or find a keyboard command set. If the core services were missing, however, the keyboard commands would be of no use.

CWSTART DLL

The main DLL that starts Hawk is named `CWSTART.DLL`. It performs numerous initialization tasks, such as loading other DLLs and reading the configuration file. The source code to `CWSTART.DLL` is provided. Many of the extension functions are defined in this DLL. To use them in your own source code you need to include the header file `CWSTART.H`, which is in Hawk's `CWSTART` subdirectory.

The tree diagram and file descriptions below will help you locate the functions provided in the various source files for `CWSTART`.

Figure 15-10. Tree Diagram of CWSTART.DLL

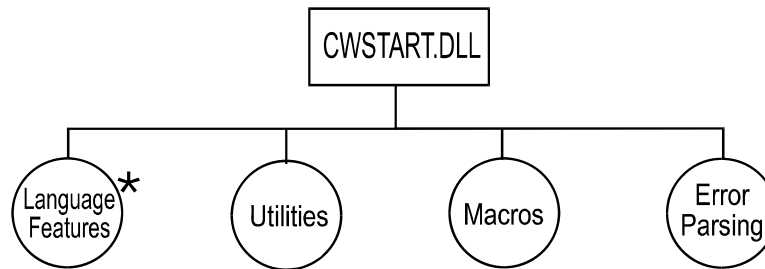


Table 15-15. Source Files

ASM.C	AUTOSAVE.C	BRACE.C	ERRORFIX.C
C.C	COMPILE.C	CURSOR.C	ERRINFO.C
CPP.C	CONFIG.C	ENTAB.C	
LANGUAGE.C	CWSTART.C	PREPROC.C	
PAS.C	DLGMENU.C	PROMPT.C	
HTML.C	FILTER.C	REPEAT.C	
CWJAVA.C	OUTPUT.C	SLIDE.C	
	STATE.C	TABSET.C	
	UTIL.C	TAGS.C	
	VCS.C	EXECMAC.C	
	WWRAP.C		
	DLGUTIL.C		
	NAMEMARK.C		
	OUTLINE.C		
	PROJECT.C		

- Support for additional languages, such as Paradox and dBASE/Clipper are contained in separate DLLs. These DLLs are Language Supplements, and their use is optional. See the topic [Supplemental Language Support](#), for a more complete description.

Descriptions of the files listed in the preceding diagram are provided in the following tables.

Table 15-16. Language Features

File	Description
LANGUAGE.C	The file that contains the engine for language specific features, such as ChromaCoding, Template Expansion, and Language Indenting. It relies on a series of functions being defined elsewhere that incorporate the corresponding filename extension into the function name (such as, <code>_c_indent()</code> , <code>_pas_indent()</code>).
ASM.C	This file contains the functions that support language specific features for assembly language source files (.ASM files).
C.C	This file contains the functions that support language specific features for C language source files (.C files).
CPP.C	This file contains the functions that support language specific features for C++ language source files (.CPP files).
PAS.C	This file contains the functions that support language specific features for Pascal language source files (.PAS files).
HTML.C	This file contains the functions that support language specific features for HTML source files (.HTML files).
CWJAVA.C	This file contains the functions that support language specific features for Java language source files (.JAVA files).

Table 15-17. Utilities

File	Description
AUTOSAVE.C	Contains the functions in support of the Auto-save feature.
COMPILE.C	Contains the functions for the Compile, Make, and Rebuild selections on the Project menu.

Table 15-17. Utilities (Continued)

CONFIG.C	Contains the functions that read, write and process the configuration file.
CWSTART.C	The functions in this file initialize the CWSTART DLL.
DLGMENU.C	Contains the supporting functions for the Popup Menu defined by CWRIGHT.MNU.
FILTER.C	Contains the functions that support the Filter selection on the Project Properties dialog.
OUTPUT.C	Contains the supporting functions for the tabbed Output Window.
STATE.C	Contains the functions that read, write and process the state file.
UTIL.C	Contains functions used by one or more of the keymaps and other shared functions, such as NextWord/PrevWord.
VCS.C	Contains the functions supporting the Version Control submenu.
WWRAP.C	Contains the function in support of the Word Wrap and related features.
DLGUTIL.C	Contains functions for enhancing Hawk dialogs.
NAMEMARK.C	Contains functions for naming Hawk's bookmarks.
OUTLINE.C	Contains functions for Hawk's Outline Symbols feature.
PROJECT.C	Contains certain functions relating to Hawk projects.

Table 15-18. Error Parsing

File	Description
ERRORFIX.C	This file contains the engine that drives the various error parsers for different compilers and languages.
ERRINFO.C	This file contains the actual error parsers for the various compilers.

Table 15-19. Macros

File	Description
BRACE.C	The source code file for the Hawk brace matching features, including the functions <code>Brace</code> , <code>BraceMatch</code> and <code>BraceMatchNext</code> .
CURSOR.C	This file contains functions to make and support mouse assignments and operations.
ENTAB.C	The functions that support entabbing and detabbing the current buffer are contained in this file.
PREPROC.C	Contains the functions in support of the Preprocess function.
PROMPT.C	This source file contains the prompting functions and supporting functions that use the prompt history facility.
REPEAT.C	Contains the source code for the function <code>Repeat</code> , which repeats a command a specified number of times.
SLIDE.C	Contains the functions for moving a block of text in or out.
TABSET.C	Contains the functions supporting setting tabs, including extension specific settings.
TAGS.C	The source file for the functions that support the CTags features.
EXECMAC.C	Contains source code for Hawk's API Macros.

CWDIALOG DLL

Many of the Hawk dialog functions are contained in a DLL that is loaded during initialization. The name of this file is `CWDIALOG.DLL`. This library contains the dialog functions that support the menuing and other interactive operations. The source code for the `CWDIALOG.DLL` is supplied Hawk.

There are a number of functions in this DLL that allow you to access the dialogs on the menu directly. It is sometimes useful to assign these functions directly to keys, or to call them from your own source code. These functions are the functions beginning with `Dlg...` and `Print`. To use these functions, just include the file `CWDIALOG.H` into your C source code. You will find this file in Hawk's `CWDIALOG` subdirectory.

CWHELP DLL

The `CWHELP.DLL` contains the functions that access the various Help libraries used by Hawk. These include the contextual help for the editor, the Hawk API function help, and

the Microsoft Windows API (SDK) help, if you have it. The name of the DLL is `CWHELP.DLL`.

If you want to call the function `cwhe1p` from your C source code, you will need to include the file `CWHELP.H` in your file. This header file is located in Hawk's `CWHELP` subdirectory.

Keyboard Command Sets

Hawk is supplied with four keyboard command sets, also called keymaps. They include the following:

- CUA (Common User Access)
- BRIEF emulation
- Epsilon emulation
- vi emulation

Each is contained in a separate DLL. The C source code files and makefiles are included for each of these DLLs. Unless you have elected not to install portions of the source code, each set of source code has been installed in its own subdirectory.



- For more information on Hawk's keymaps, refer to the topic [Using Keymaps](#) in [Chapter 12 Customizing the Interface](#).

Supplemental Language Support

In addition to the support for C/C++, Pascal, HTML, and assembly language built into the `CWSTART.DLL`, support is provided for other languages in separate DLLs. The source code for these DLLs is provided. Below is a description of the language support DLLs provided:

Table 15-20. DLL Language Support

DLL	Description
PRG.DLL	This DLL contains the functions that support language specific features for dBASE and Clipper language source files (.PRG files).
SC.DLL	This DLL contains the functions that support language specific features for Paradox PAL source files (.SC files).
COB.DLL	This DLL contains the functions that support language specific features for COBOL source files (.COB files).
BAS.DLL	This DLL contains the functions that support language specific features for Visual Basic for Windows (.BAS files).

Table 15-20. DLL Language Support (Continued)

CWHTML.DLL	This DLL contains additional support for the HTML language, including a split window viewer.
CWJAVA	This DLL contains the functions that support language specific features for JAVA source files.

These DLLs are not automatically loaded, as is `CWSTART.DLL`. You will need to use the `Libraries` dialog on the `Tools -> Customize` menu. It adds a statement to your Hawk configuration file (`MWHAWK.INI`). Below are example load statements for these DLLs:

```
[LibPreload]
LibPreload=PRG.DLL
LibPreload=SC.DLL
LibPreload=COB.DLL
LibPreload=BAS.DLL
```

The statements above load the DLLs and make the functions in them immediately available for use. This does have the effect of slowing the initial loading of Hawk. An alternative is to use `LibAutoLoad` rather than `LibPreload`. This will cause the DLLs to be loaded only on demand. It does, however, require listing all of the function in the DLL in the statement. See the description of this function for details.

Auxiliary Services

The Auxiliary Services or DLLs provide support for supplemental features. These features or groups of features are generally provided in separate DLLs. The source code for each DLL, if provided, is in a subdirectory within the Hawk home directory. Each subdirectory is given a name that is the same as the root name of the DLL.

These `Features` or DLLs are Add-On packages that may not be required by every user. Like the additional language support DLLs, they may require a little setup to use. They are generally not loaded automatically.

Table 15-21. Additional Language Support DLLs

DLL	Description
CWLDRAW.DLL	This DLL allows remapping the keyboard so that you can draw lines and boxes in your file, using the IBM OEM character set. You must be using <code>OEMFixed</code> , <code>8514oem</code> , <code>terminal</code> , or similar font. If not, these lines will be displayed as international characters. It is not loaded automatically.

Table 15-21. Additional Language Support DLLs (Continued)

CWREDIT.DLL	This DLL provides the user with the ability to edit the <code>Tool</code> <code>ToolBar</code> and <code>SideBar</code> interactively. It supports the <code>Tools -> Customize -> Toolbars</code> menu selection.
CWDIFF.DLL	This DLL contains the support for the Hawk differencing feature. It allows side-by-side differencing of files or buffers, in addition to single buffer difference analysis.
CWFOPEN.DLL	This DLL supports the <code>File Open</code> mode of the Hawk <code>SideBar</code> .
CWDDE.DLL	This DLL contains the functions that turn Hawk into a DDE server. Other applications can then operate Hawk remotely to perform syntax checking, Lint, and Tags database updating. The DLL is not loaded automatically.
CWBROWSE.DLL	This DLL contains the main support for the Hawk Browser. This DLL is loaded automatically, when needed.
CWTAGS.DLL	This DLL supports the use of Starbase compiled Tags databases with the Browser. It is loaded as needed.
CWBSC7.DLL	This DLL supports the use of Microsoft Browser databases (C/C++ V7.0+) with the Browser. It is loaded as needed.
CWWEB.DLL	This DLL supports loading files edited in Hawk into various Web browsers. It is not loaded automatically.

Sample DLL

As mentioned, the source code for an essentially empty sample DLL is included with a full Hawk installation. You will find it in the `SAMPLE` subdirectory in the Hawk home directory. You can add to this file, or you may use it as an example of what elements go into a Hawk DLL.

The source code for `SAMPLE.DLL` is a skeleton of comments and function stub or two. The comments show you where to insert additions or changes that you want to make to Hawk. These insertions might include some of the following:

- Loading other DLLs, possibly written in a programming language other than C.
- Adding event handlers.
- Setting defaults, much as you would in the configuration file.
- Adding key commands.
- Writing and installing replacement functions.

A makefile is also included for this DLL.

Dissecting a Hawk DLL

Your compiler imposes certain requirements in order to correctly produce a Microsoft Windows-compatible DLL. In addition to those requirements, there are a few essentials required by Hawk. This section will help you understand what Hawk requires and why. The following sections describe a typical Hawk DLL.

The `_init` Function

Whenever Hawk loads a library, it executes that DLL's `_init` function, if the library has one. This is the function you use for initializing the DLL and letting Hawk know what functions the library has to contribute. Generally, this function consists of a series of `LibExport` function calls, which export the functions defined in that DLL for use by other entities in Hawk.

Exporting Functions

The functions in a DLL, and the parameters for each, must be registered with Hawk before you can make full use of them. If you don't register them with Hawk, you will not be able to assign them to a key or call them from the `API Command Key`. You register or export the functions with the `LibExport` function. This is in addition to any export declarations required by the programming language.

This requirement is a result of the use of Pascal calling conventions. When using Pascal calling conventions, the program stack would become unbalanced if any function parameters were omitted. Yet, the function `LibFunctionExec`, which services the `API Command Key` and the key assignments, has no way of knowing the type and number of parameters are required unless you tell it. The `LibExport` function does this.

- If you change the number or type of parameters in a DLL function, you must remember to change the corresponding `LibExport` call that registers that function.

Making Changes and Additions

When you have determined that you are going to modify the Hawk source code, you have two approaches available to you. You can change existing functions or add new functions to the system.

Changing Existing Functions

Changing an existing function is often an efficient method of change, if your change is small and you want the change to be reflected whenever a certain function is called. This usually avoids the need to create or change a `LibExport` call.

For example, you might want to change the function that moves the cursor ahead in increments of a word. You can be relatively certain that this function is going to be called as the result of some key command.

If you were less certain about what other routines might be using the function, you would need to consider the effects of your changes on those other routines. If you are in serious doubt about what routines may be relying upon the function, it is best not to change it. Instead, create a new function that is called when you want it, and leave the old function in place for the routines that use it.

Adding Your Own Functions

When adding your own functions, you must be sure that they are declared in the same manner as other Hawk functions. You won't be able to assign your functions to keys, call them from the API Command Key, and other useful things, unless you properly export your functions to Hawk. This means using `LibExport`.

If you create a new file for your functions, you will have to be sure it uses the necessary header files (`CWSTART.H`, `CWDIALOG.H`, `CWHELP.H...`), and that it is included in the compile and link.

There is a macro defined in `EXPORTS.H` that simplifies defining new functions for Hawk, when programming in C. Just define your function as `DLL` and it is automatically declared as Pascal calling conventions, exported, and so forth. Here is an example of its use:

```
void DLL
KeyNotInUse( void ) {
    SysBeep();
    MsgWarning( "Unassigned key" );
}
```

Windows NT also requires that all functions be added to the `.DEF` file in order to be properly exported. Just edit this text file and add your function name to the end of the existing list. Also note that Windows NT is case sensitive in function names, whereas Windows 95 and Windows 98 are not.

Creating New Keymap Command Sets

Keymaps have additional requirements over and above those of a Hawk DLL. This section describes the basic outline of a keymap.

Keymap `_init` Function

Like the `_init` function for any other Hawk DLL, the Keymap's `_init` function typically contains a series of `LibExport` calls that make functions available to Hawk. This is necessary for any functions you create to assign to keys within the keymap.

Normally, the `_init` function does not install the keymap. If it did, you would not be able to load the DLL to have access to its functions without loading the keymap.

Keymap Function

The keymap function is the function that creates and installs the new keymap. To be compatible with the `DefaultKeymap` function, this function must have the same

name as the root of the DLL in which it resides. For example, the CUA keymap is in the file `CUA.DLL` and is installed with the `CUA` function.

Flag Initialization

One of the first tasks in your keymap function is to set options the way users of your keymap will expect to find them. For example, the `BRIEF` keymap needs to allow assignments to `Alt` keys, and does not create a new window for each new file that is loaded. If the users of your keymap expect to have vertical and horizontal scroll bars on their edit windows, now is the time to implement that functionality.

There are several functions that assist in setting the options you require. They are `SysSetFlags`, `SysSetDefault`, and `SrchSetFlags`:

- Use `SysSetFlags` to set options you might otherwise set from various Hawk dialogs.
- Use `SysSetDefault` to set up window and buffer settings the way you want, even before any buffers or windows have been created.
- Use `SrchSetFlags` to set the default search settings.

You are not dictating with these settings how a user must operate. The settings you create here may be overridden by settings stored in the configuration file or state file.

Basic Assignments

When you create a new keymap, it is empty. The only keys or mouse actions that will work are those that Hawk lets Windows process. This includes menus, `HT`, `HX`, `H$` and such. If you want pressing the `A` key to produce an `A` in a buffer, you need to make that key assignment.

There are functions provided to take the drudgery out of making these assignments. `KmapAssignTypables` and `KmapAssignRange` are your primary assistants. The difference between these two functions is that the first makes assumptions about what you want the keys to do, whereas the second allows you to specify the function that is assigned to the keys.

Keymap Specific Assignments

Now you are ready to make assignments specific to your keymap. Your main tool in doing this will be the function `KmapAssign`. Use the Hawk API functions, or create your own supporting functions to assign to keys. Remember that when you create your own functions you will need to export it with a `LibExport` call in the DLL's `_init` function in order for the key assignment to work. Otherwise, you will get a "function not found" message when you press the keystroke.

Menu Accelerators

The last step in creating your own keymap is to put strings indicating any "shortcut" keys or accelerators on the menu. You will rely on the `MenuAddKeyString` function to do this.

Recompiling a DLL

Part of the Hawk directory structure is a miniature of that used by most C compilers. There is an `INCLUDE` subdirectory that contains header files, a `LIB` subdirectory to contain `.LIB` files, and subdirectories to contain the source code for each of the Hawk DLLs you can rebuild. The `INCLUDE` and `LIB` subdirectories must be made known to your compiler and linker. You can do this manually, or through the use of a `MAKE` utility or `Project` file, if your compiler supports these things.

Using and Modifying the Makefiles

There are two types of makefiles supplied for each of the DLLs that can be modified. Only one type is installed for each subdirectory containing DLL source code. The type installed depends on the choice made at installation. The choice is between Microsoft or Borland makefiles. If you did not choose a "Full" Hawk install, you would not have been offered the choice of Microsoft or Borland source code, and they would not be installed.

The primary things that makes these files specific to one compiler or another are the compile and link command lines that they employ. If you are using some other compiler, you should find adapting it to your compiler is a straightforward task. You can recognize the Microsoft makefile by looking for the file with no extension that has the same root name as the DLL it makes. The Borland makefile uses the same name, except that the extension used is `.MAK`. Once again, if the desired makefile does not appear, do a "Full" reinstall of Hawk, and choose the appropriate option when the prompt comes up.

The compiler needs to know that you intend to create a 32-bit DLL. Defining the environment variable `CPU` is normally sufficient to cause the Microsoft makefile to generate a 32-bit DLL. Alternatively, you may generate a 32-bit DLL for Borland or Microsoft by defining a macro named `WIN32` at the beginning of the makefile:

```
WIN32=TRUE
```

The DLLs themselves are generated in a subdirectory of the source file directory. The name of the directory varies depending on source type (Borland or Microsoft) being compiled. Usually it will be something like "Release" or "Rel". Once the DLL has been compiled, it is advisable to move it up to the Hawk executable directory, although this is not required.

The Borland compiler often relies on configuration files and command line switches to locate include files and libraries, rather than environment variables. As a result, it may be necessary to modify a macro definition in the Borland makefile to indicate the location of these files, unless you have installed the Borland compiler in the default directory. If you have installed your Borland compiler in another location, modify the value assigned to `BORPATH` toward the beginning of the makefile. For example, if you installed the compiler in the directory `D:\BC`, change the assignment to read as follows:

```
BORPATH=D:\BC
```

Adding Files

Groups of related files have been defined as macros toward the beginning of each makefile. This facilitates operating on the filenames as a group. Also, if you choose to place the source code for some extensions to Hawk in a separate file, you can readily add those filenames to the macro definitions. If the makefile employs a linker response file, such as `CWSTART.LNK`, don't forget to add your file to the list in the response file so that it will get linked in.

Compile and Link Options

The compiler options you use for compiling Hawk DLLs are largely the same as you would use for compiling any Microsoft Windows DLL. Hawk DLLs must be compiled using `Large Memory Model`. If you specify your include directories on the command line, be sure to include the Hawk `CWSTART` and `INCLUDE` subdirectories in your command. You will find that this has been done for you in the makefile.

Similar requirements apply to your `Link` command. Begin with your basic `Link` command line for producing a Windows DLL. The libraries in Hawk's `LIB` subdirectory must be linked with the other objects. A standard `.DEF` file has been supplied for each DLL.

Link Libraries

The makefiles provided reference certain libraries in the linker command. You may find it useful to know the purpose of each of these libraries in case you have a different version of the same compiler or would like to construct a similar command line for a compiler not supported.

Microsoft Link

This is the command line in the makefile for Microsoft C invoking the linker:

```
link /NOD $(LIBDIR)\libentry @cwstart.lnk, $(DLLDIR)\
cwstart.DLL, ,libw oldnames\ ldllcew $(LIBDIR)\
cwright,cwstart.def
```

The libraries used in this link command are `LIBW.LIB`, `OLDNAMES`, `LDLLCEW.LIB` and `CWRIGHT.LIB`. Of these, the files `LIBW.LIB` and `CWRIGHT.LIB` are import libraries. They don't contain any object code themselves, but rather tell where the routines may be accessed at run-time. `LIBW.LIB` points to addresses in `USER.EXE`, `GDI.EXE` and other executables that provide the Windows API. `CWRIGHT.LIB` points to addresses in the Hawk kernel that support the Hawk API.

You may need to link in other import libraries at times, for example, if you are making direct calls to routines in `CWSTART.DLL`, `CWMATCH.DLL` or a DLL of your own creation. If you don't have an import library (`.LIB`) file corresponding to the DLL you are using, you can create one with the `ILIB` utility supplied.

LDLLCEW.LIB

The library `LDLLCEW.LIB` contains object code to use when creating Large Memory Model Windows DLLs. Depending on your installation of Microsoft C, you may find that

your Large Memory Model library is named `LDLLCAW.LIB` instead. Alternately, you may find that you have not installed any large model library at all. If so, you can run the setup program that came with your compiler to add to your installation. If you use a library intended for any other memory model you are guaranteed to have a program that doesn't work right, and will probably crash Windows.

OLDNAMES.LIB

The library `OLDNAMES.LIB` is included to avoid unresolved externals. This is because Microsoft changed the names of some of their library routines. See your `MSC 7.0 README` file for details. If you are using an older version of the Microsoft compiler, you should remove this library from your command line.

Borland Link

Below is a portion of the input script for `TLINK` in the makefile for Borland C/C++:

```
..\lib\cwright.lib+
import.lib+
cwl.lib
```

The purpose of these libraries follows closely the libraries described above for Microsoft. The libraries `CWRIGHT.LIB` and `IMPORT.LIB` are import libraries. `IMPORT.LIB` is supplied to you by Borland for the Windows API and `CWRIGHT.LIB` comes from Starbase for the Hawk API.

You may need to link in other import libraries at times, for example, if you are making direct calls to routines in `CWSTART.DLL`, or a DLL of your own creation. If you don't have an import library (`.LIB`) file corresponding to the DLL you are using, you can create one with the `ILIB` utility supplied.

The third library `CWL.LIB` contains object code to use when creating Large Memory Model Windows DLLs. This library was named `CWINL.LIB` in versions prior to 3.0. Use only a Large Memory Model library intended for Windows for this purpose.

Using Your Own DLL

There are several reasons that you may have for wanting to use a DLL that you have developed on your own, rather than modifying one of those supplied. It is expected that if you write a new command set for the keys that you will place it in a separate DLL. You can then use the `DefaultKeymap` function to facilitate loading the DLL and calling the keymap subroutine that makes the key assignments.

You may find it more convenient to develop a DLL with Borland Pascal for Windows, or Microsoft's Visual Basic than to modify an existing C language DLL. Alternately, you may just be concerned about how well your C language additions will coexist with the source code provided.

Your DLL will need all of the same parts described in the "Dissecting a Hawk DLL" section above, regardless of what language you use to produce it. Be sure to compile using a Large Memory Model. You need Far calls and pointers for things to work properly.

Installing Your DLL

When you are ready to use your DLL, you need only reference it in the proper section of your configuration file. If your DLL is a keymap and supporting functions, just change the name assigned to `DefaultKeymap` to the basename of the DLL file; if your DLL is named `MYKEYS.DLL`, change the statement to read `DefaultKeymap=mykeys` under the `[DefaultKeymap]` section.

If your DLL just contains some additional subroutines you want to have available, you have a couple of options:

- Always load the functions at startup-time,
- OR -
- Tell Hawk to load them when needed.

The first option may slow down the initial loading of Hawk, whereas the second can cause a short pause when the function is first called. The second option may be better if you have a nominal amount of memory installed. If you have a fast CPU and hard disk, you probably won't notice much difference in daily use.

The following examples assume that you have put your DLL in Hawk's home directory, or in a directory that your `CWLIB` environment variable points to. The first loads the DLL immediately, while the second loads it only when needed. The first option is done automatically when the library is loaded in Hawk's `Customize -> Libraries`. Loading the library in this dialog adds the following line to the `[Editor]` section of the Hawk configuration file (`MWHAWK.INI`).


```
LibPreLoad=myDLL
```

- The second option, loading the DLL only as needed, must be done manually by placing the following commands in the configuration file under the `[Editor]` section.
- `LibAutoLoad="myDLL func1 func2"`

This option requires that you supply the names of the functions in the DLL which may be called.



- For more information on configuration files, refer to [Chapter 17 Configuration Files and Command Line Parameters](#).



Hawk operates on the Intel platform under Windows 95, Windows 98 and Windows NT. Hawk does not operate under the UNIX environment. There are some Hawk options, however, that allow UNIX files to be edited within the program.

The following sections are included in this chapter:

[File Type and End-of-Line \(EOL\) Characters](#)

[Compiling UNIX Programs from Hawk](#)

[Filename Case and File Securities in UNIX and Windows.](#)



- Hawk does not provide the utilities needed for setting up NFS between UNIX and Windows systems. They must be obtained from third party vendors.
- UNIX files can be loaded via NFS (Network File Systems) networking, or through FTP (File Transfer Protocol). Consult a network administrator to find out about the possibilities for networking the Windows and UNIX systems via NFS.

File Type and End-of-Line (EOL) Characters

UNIX style end-of-lines characters are used for working in an environment that requires transferring files back and forth between a DOS and UNIX system. In this case, you may wish to have Hawk preserve the native UNIX End-of-Line (EOL) characters.

Save File as New Type

The `File -> Save As` dialog allows you to convert UNIX files to a "Macintosh Text" or "MS-DOS Text" format. Conversely, other file types may sometimes be converted to "UNIX Text" format. In the Save File as Type or Save as Type combo box, look for file conversion possibilities following the normal filter entries.

Original File Type	Possible Conversions
Unicode	"Normal Text"
UNIX	"Macintosh Text" "MS-DOS Text"
Macintosh	"UNIX Text" "MS-DOS Text"
Otherwise	"Unicode Text" "UNIX Text" "Macintosh Text"



Macintosh, Unix and DOS option are only available if Auto-Sense File EOL is marked in `Customize -> Environment -> General`; Unicode is only available if Auto-Detect Unicode Files is marked on the same dialog.

Make UNIX EOL Characters the Default

DOS style end-of-lines are composed of a carriage-return and line-feed, and are the initial default setting in Hawk. UNIX end-of-lines are a line-feed character only. To enable the UNIX end-of-lines as the default, complete the following steps:

1. Select the `Customize -> Language` dialog.
2. Click the `Options` tab.
3. Highlight all applicable file type extensions.
4. Check the `UNIX EOL` check box.
5. Press `OK` to save the setting.

Making this selection does not change end-of-line sequences already in the buffer. It affects only those inserted after the selection is made.

Enable Auto-sense File Type Option

You will also want to enable the `Auto-sense File Type EOL` option. When selected, Hawk will look for line terminators when it first loads a file. It can detect DOS (`cr/lf`), UNIX (`lf`), and Macintosh (`cr`) line terminators. If the file has UNIX style line terminators, the `Enter` key is made to insert only a line feed (`lf`). Macintosh translations for both input and output are done at a low level and seem like DOS files to the user.

To select `Auto-sense File Type EOL`:

1. Select the `Customize -> Environment dialog`.
2. Click on the `General` tab.
3. In the `File Loading/Reloading` group, check the box `Auto-sense File Type EOL`.

Specify UNIX EOL Characters on a File Type or Per File Basis

You can also specify the type of end-of-line characters that are inserted into a new file (DOS, Unix, or Mac), and even choose to convert existing characters to the new type. These EOL options can be found in the following locations:

- On the `Options` tab of `Customize -> Language`, to specify an EOL style on a file-type (e.g. `.C`, `.TXT`, `HTML`, etc.) basis.
- On the `Options` tab of `Document/Window Manager` dialog, to specify an EOL style on a per-document basis.

After making the setting change for the EOL type, each file that belongs to the selected file type in `Customize -> Language -> Options`, or that is selected in the `Document/Window Manager`, is scanned to see if it contains inappropriate EOL characters. If found, a dialog prompts you with the opportunity to change the EOLs.

EOL characters are converted as follows:

File Type	Characters	"C" Escape Characters	Hex Chars	Regex Chars
DOS	CR (carriage return), LF (line feed)	<code>\r\n</code>	<code>0x0d, 0x0a</code>	<code>\x0d\x0a</code>
Unix	LF	<code>\n</code>	<code>0x0a</code>	<code>\x0a</code>
Mac (file)	CR	<code>\r</code>	<code>0x0d</code>	<code>\x0d</code>
Mac (document)	LF	<code>\n</code>	<code>0x0a</code>	<code>\x0a</code>

For the Macintosh file type, the EOL characters will appear just the same as for Unix (just a line feed); Hawk converts the CR to the LF as it reads the file, and converts the LF to the CR as it writes the file. If you search through a Macintosh file in hex mode, you will therefore find Unix style 0x0a characters instead of Mac style 0x0d characters. .



Also, saving a Unix file as MS-DOS Text or Macintosh Text has the same effect as converting the EOL characters within the file, except that the file is also saved under a new name. Refer to the preceding topic on [Save File as New Type](#), in this chapter.

Change EOL Characters in the Source File

EOL characters in your source file can be changed by performing a simple Search/Replace operation. To do this, search for one type of EOL character and replace it with another.

The Hexadecimal Values for End of Line Characters include the following:

```
0D = Macintosh
0A = UNIX
0D0A= DOS
```

Although you can search for a hexadecimal value (`\x0A`), it is better to use a regular expression for this operation..



See the topic [Searching for Control Characters \(binary/hex data\)](#) in [Chapter 10 Search and Replace and Other Navigational Tools](#).

Examples

To change DOS EOLs to UNIX EOLs, complete the following steps:

1. Mark the `UNIX` radio button in the `EOL` group for the appropriate file type in `Customize -> Language -> Options`
2. Select `Search -> Replace`.
3. Check the `Regular Expression` box.
4. Search for `\n` and replace with `\n`. (The meaning of the `\n` is a new line (`cr/lf` or a `lf`.)

Because the Macintosh EOL is a (`cr`) only, it is better to use `\r` when searching or replacing it.

To change DOS EOLs to Macintosh EOLs, complete the following steps:

1. Go to the `Search` menu.
2. Select `Replace`.
3. Check the `Regular Expression` box.
4. Search for `\n` and Replace with `\r`.

Macros for Automating UNIX EOL Conversion

In addition to the available method for manually searching and replacing UNIX/DOS EOLs, Hawk provides an API macro that performs the conversion automatically. API macros are described in [Chapter 15 Extending Hawk](#). To use the macro, do the following:

1. Click **API Macros** on the **Tools** menu.
2. In the API Macros **Edit** dialog, choose the **UNIXEOLs** macro from the drop-down list under the **Name** edit box.
3. Run the macro using one of the following methods:
 - Click **Run** in the API Macros **Edit** dialog.
 - Click **Close**, then click **Run API Macro <UNIXEOLs>** on the **Tools** menu.

The macro will search the current document for all EOLs, and replace them with UNIX EOLs.

Compiling UNIX Programs from Hawk

Hawk executes compile commands by creating a DOS shell and executing the command, as though it were being run at a DOS prompt. The output of the compile/build is then captured and presented to you in the Hawk **Output** window.

If you compile UNIX programs, you will need to do this outside of Hawk. One possible solution to accessing your UNIX system from within Hawk is to use a Telnet implementation to issue commands automatically from a file.

Example

Set up the file with the necessary commands to do a build. This allows you to click the **BUILD** button in Hawk to launch a Telnet session, which issues the necessary commands. Then Hawk parses the output and steps through the errors.

- Another solution:
Use **RSH** (remote shell), which allows one computer to run a command on the other. **RSH** comes with NT and works just like the UNIX **RSH** command. Use it to control a batch UNIX process, like **MAKE**, from Hawk via an NFS mounted disk drive. **RSH** along with **FTEE** make the perfect fit.

Although Microware cannot assist in setting up access to UNIX systems, we can help make the necessary customizations to the **BUILD** command, given that the **Build** or **Compile** command can already be performed successfully from a DOS prompt.

Filename Case and File Securities in UNIX and Windows

Filename case can sometimes be lost when saving files to UNIX systems from Hawk. If you want the same kind of case-sensitivity you would have on a UNIX system, check the option **Allow Case Sensitive Filenames** on the **Customize -> Environment -> General** tab.

The loss of filename securities can also be an issue using the traditional file-saving method in Hawk. To preserve file securities on UNIX files, use the Hawk alternative file-saving method, called `File Rewrite Save Method` (also on the `General` tab).

To select these options, complete the following steps:

1. Select the `Customize` -> `Environment` dialog.
2. Click on the `General` tab.
3. Check the options `Use file rewrite save method`, and `Allow Case Sensitive Filenames`.

17

Configuration Files and Command Line Parameters

This chapter includes the following sections:

[Configuration and State](#)

[Command Line Parameters](#)

Configuration and State

The following sections describe the location and contents of configuration and state files within Hawk.

Location of the Configuration File

Hawk configuration data is stored in the file `MWHAWK.INI`. During initial installation `MWHAWK.INI` is placed in the same directory as the Hawk executable file. You can move this file to another location, rename the file, and maintain several configuration files in different work areas.

If the Hawk executables are installed on a network, it is recommended that you place your configuration file(s) in a separate location. This maintains the integrity of individual configurations.

Upon starting, Hawk searches several locations to determine what configuration file to use. When Hawk determines a path to the location of the configuration file, or locates the file itself, it stops searching and proceeds to read the file.

Following are the locations in which Hawk searches for the integration file, in the order of searching:

1. The command line for a specified `/C` parameter
2. The environment settings for a variable named `MWHAWKINI`
3. The working directory for a file named `MWHAWK.INI`.
4. The directory containing `MWHAWK.EXE` for the file `MWHAWK.INI`.
5. The Windows directory for the file `MWHAWK.INI`.

The command line parameter, or the environment variable, will have an associated string value when it is used to point to the configuration file. This string names the directory in which the configuration file resides. It may also name the file itself.

Examples

The string may take the following form when specifying a directory:

```
d:\source\project1
```

It may also take the following form when specifying a file:

```
d:\source\projects\cencom.cfg
```

If Hawk finds no configuration file, one will be created in the working directory.

- Hawk can function without a configuration file, but will notify when one is not found.

Introduction to Configuration and State

To understand Hawk projects and workspaces, it's helpful to know how Hawk maintains its configuration and preserves its state between sessions. This helps determine which

information to keep in which file. Understanding configuration also helps when modifying the configuration file directly.

Configuration File

The configuration file contains your Hawk settings. The settings can be specified during installation, modified in dialog boxes, or by directly editing the configuration file. In the latter case, the configuration file might contain almost any Hawk commands. The configuration file is named `MWHAWK.INI`.

The configuration file is a text file that follows the same general format as other Windows `.INI` files. It contains section headings enclosed in square brackets, followed by configuration statements.

- Headings that appear in a Hawk configuration file include the following: `[Editor]`, `[LibPreload]`, `[Colors]`, `[DefaultKeymap]`, `[KmapAssign]`, `[Printer]`, `[Compiler]`, `[VersionControl]`, `[Definitions]` and `[Fmatch]`.
- The configuration statements contain keywords with string assignments. For the lines containing keywords, the format is as follows:

```
<function>=<param 1>[,<param 2>,...<param n>]
```

The following criteria apply:

- The keywords are actually the names of functions in the Hawk API.
- The string assigned to the keyword contains the parameters required by that function.
- The assigned string can contain white space.
- The parameters can be numbers, strings, constant identifiers and operators.
- Variables and nested function calls may not be used as parameters.

State File

The State file contains information about the buffers and windows open the last time Hawk was exited. It also contains the more transient information about your Hawk settings, such as your search options and responses to prompts. It follows the same rules as a configuration file, but it is normally limited to one section named `[state]`. This file is named `MWHAWK.PST`.

Other Files Containing Configuration Data

Project files also contain configuration information. These files are intended to be swapped on the fly, whereas the configuration file remains constant throughout a session.



The project file overrides any information duplicated in the configuration file.

As settings are changed in the various dialog boxes, the configuration is automatically updated. You can select which file the dialogs use to update configuration information:

the configuration file or the project file. You can further select which categories of information are kept in which file.

Following is the main distinction between the two methods of storing configuration information:

- Configuration information stored in the project file is associated with a specific project.
- Configuration information stored in the configuration file is associated with multiple projects.

Example File

Below is an abbreviated example of a configuration file:

```
[DefaultKeymap]
DefaultKeymap=BRIEF

[KmapAssign]
KmapAssign=<Ctrl-`>, Preprocess

[LibPreload]
LibPreload=PRG.DLL

[Editor]
KeyDelay=1200
KeyRepeat=5
Autosave=20
SysSetFileLocking=30

[Colors]
ColorError=0xe0
ColorWarning=0xd4
SysSetDefault=DEFAULT_COLOR_TEXT,0x1f
SysSetDefault=DEFAULT_COLOR_SELECTION,0x9e

[Printer]
PrintFooter="-Page %p-"
PrintHeader="%f %d %t"
PrintFlags=145
PrintMarginBottom=0
PrintMarginRight=0
PrintMarginLeft=0
PrintMarginTop=0

[Definitions]
EvalStrAdd=DEBUG,1
```



```
[Compiler]
CompilerAssign='Borland C++','.c'
```

Example Interpretation

Following is an interpretation of the contents of the example configuration file above. The analysis proceeds section by section:

[DefaultKeymap] Section

The default key command is set to `BRIEF`. This causes `BRIEF.DLL` to be loaded and its function `BRIEF` to be called.

[KmapAssign] Section

In the `[KmapAssign]` section, a key assignment is added to the default keymap. The `Preprocess` function is assigned to the `C Backquote` (```) keystroke.

[LibPreload]

One of the supplemental language support DLLs is loaded for use.

[Editor] Section

The `[Editor]` section of the example configuration file sets the keyboard delay and repeat rate. Auto-save is set to occur after 20 seconds of keyboard inactivity. File locking is enabled for 30 file handles.

[Colors] Section

The `[Colors]` section of the example sets the color of error messages to black on yellow, and the color of warning messages is set to red on gray. These are global settings. The next two color settings may be set differently for each edit window. For this reason, we do not use the functions `ColorText` and `ColorSelection` to set the colors, but rather set what the default color will be. You use the `SysSetDefault` functions when setting defaults for things that are specific to individual buffers or windows. In this case, the color of text is set to white on blue, and the highlight for the selection is set to yellow on light blue.

[Printer] Section

The `[Printer]` section and a number of other sections use private, undocumented functions. The `Print` menu item within Hawk controls the contents of this section. It is not recommended that you modify the `[Printer]` section.

[Definitions] Section

This section defines the label `DEBUG` and gives it a value of one. This label may be used in numeric expressions processed by Hawk, including those given through the Command Key and the `Preprocess` function (`#ifdefs`).

[Compiler] Section

In this section, the Borland C++ compiler is assigned for use on files that have the `.c` file extension.

All of the sections discussed above contain configurations that can normally be set from within Hawk. In most cases, it is not necessary to directly edit `MWHAWK.INI`.

Processing At Startup

When Hawk is started, the command line is processed before the configuration file or the state file. This enables the command line to control the location of these files. It also results in the potential for the configuration file to reverse the effects of a parameter specified on the command line.

The following sections of the configuration file are automatically read at startup, in the order given:

- [Menu]
- [DefaultKeymap]
- [KmapAssign]
- [LibPreload]
- [Editor]
- [VersionControl]
- [Template]
- [Colors]
- [Ribbon]

After the portions of the configuration file that are automatically read have been processed, the state file is processed. This gives settings in the state file priority over similar commands in the configuration file.

Order of Processing

The functions under each of the section headings are executed in the order they are listed. Regardless of the order in which the sections occur, however, the sections are always executed in the same order. The [Printer], [Compiler], [Definitions] and other sections are processed as needed.

Each section heading represents a logical grouping of functions, as shown in the following list:

Table 17-1. Section Headings and Functions

Heading	Functions
Colors	Functions to set the colors for text, messages and so on. This is done with the <code>SysSetDefault</code> function.
Compiler	Used primarily by Hawk for saving associations between file extensions and compilers.

Table 17-1. Section Headings and Functions (Continued)

DefaultKeymap	The function that sets the initial key command set, usually <code>BRIEF</code> or <code>CUA</code> . For example, the statement <code>DefaultKeymap=BRIEF</code> will cause Hawk to load <code>BRIEF.DLL</code> and call its <code>keymap</code> function, which it assumes to be the same as the filename root – <code>BRIEF</code> .
Definitions	Defines labels for use in expression evaluation.
Editor	System-wide functions, default settings (except for keymap), and any other functions that are not order-dependent.
Fmatch	Used by Hawk for saving the parameters used by the <code>File Find</code> and <code>File Grep</code> features.
KmapAssign	Makes key assignments that are additions or modifications to the default keymap.
LibPreload	Loads DLLs that are not automatically loaded.
Menu	Defines the menu structure, if other than the default. The results of using the Menu Editor are placed here.
Printer	Used by Hawk for defining print margins and headers.
Toolbar	Defines modifications to the default toolbar and sidebar configuration after using <code>Toolbar Configuration</code> dialog.
Template	Contains user-defined Language Template definitions.
VersionControl	For defining command lines for <code>Check-in</code> , <code>Check-out</code> and several other related commands. The section is duplicated for each version control system defined, so that you may have separate commands for each.

Any Hawk function can be placed under any of the section headings. However, grouping functions logically under section headings helps ensure that the functions are executed in their proper order. For example, you could place the default keymap in the `[Editor]` section, but then the modifications and additions made in the `[KmapAssign]` section would be lost when the keymap is loaded over the top of them.

User-Defined Sections

Other section headings can be defined and other editor commands can be placed under those sections. These commands, however, are not processed automatically at startup.

Many functions you might want to create for assignment to a keystroke are a simple sequence of functions to execute consecutively. Return values are not examined, and no control structures are required. In most cases such tasks can more easily be accomplished with the creation of a macro. However, they can also be done with user-defined sections in the `MWHAWK.INI`. The sections can subsequently be assigned to a keystroke in Hawk.

Several examples of user-defined sections follow.

Change Search Direction

```
[ISearchBack]
SrchSetFwd=FALSE
ISearch=
SrchSetFwd=TRUE
```

This example simplifies the process of performing an incremental search backward in the buffer. Usually, the search options are set to search forward. You could bring down the menu, change the options, and then change them back, after performing your incremental search. The example above changes the search options to search backward, performs the incremental search and then sets the search direction to forward.

After you have created this section in your configuration file, you could make a key assignment like the one below, also in your configuration file:

```
[KmapAssign]
KmapAssign="<F12>", "ConfigFileRead ' ' 'ISearchBack'"
```

The previous key assignment causes Hawk to do a backwards incremental search when + is pressed. You can use similar key assignments to implement other examples or sections of your own. Most key assignments can be made more easily using the Hawk Customize -> Keyboards dialog.



For more information about assigning keys through Hawk's interface, see [Chapter 12 Customizing the Interface](#).

Save and Restore Position

The following example can be used in place of the standard save and restore position commands:

```
[RotateMarks]
MarkRestorePos=
MarkSavePos=
MarkRotatePos=
```

```
[StackMarks]
MarkSavePos=
MarkRotatePos=
```

- By using `[RotateMarks]` instead of the restore position command, restored positions are not lost, but rather moved to the other end of the list. You can then continuously cycle through your saved positions.

- If you use `[StackMarks]` to save your current position, you will restore positions in the same order you saved them instead of the reverse order in which you saved them.

Scrap Buffers

This example uses the Hawk multiple scrap buffers. To work appropriately, you must first define more than one scrap buffer. To do this, enter a value of two or higher in the `Number of Scrap Buffers` field on the `Customize -> Environment -> Clipboard` dialog.

Use the following sections instead of `Cut`, `Copy` and `Paste`:

```
[RingCopy]
BlockCopy=
ScrapNext=
MsgMessage="Copied to current scrap"
```

```
[RingCut]
BlockCut=
ScrapNext=
MsgMessage="Cut to current scrap"
```

```
[RingPaste]
ScrapPrev=
BufInsertScrap=
MsgMessage="Current scrap inserted"
```

When using these three sections in place of `Copy`, `Cut` and `Paste`, you get a last-in, first-out ring of scrap buffers. Successive `Copy` or `Cut` operations (without an intervening `Paste`) do not overwrite each other until you run out of scrap buffers. Successive `Paste` operations let you paste the contents of each scrap buffer in the reverse order they were used. The same functionality can more easily be achieved using the auto-increment option for scrap buffers in the `Customize -> Environment -> Clipboard` dialog.

Relating Check Boxes to Functions

There is a `System Options` check box in the `Directories` tab of the `Project -> Properties` dialog, and several more check boxes in the `Read Configuration Data from a File` dialog (to access the latter, click [Read](#)

Configuration Data on the Customize menu.) The following list will help you relate those check boxes to the contents of your configuration file or project file:

Table 17-2. Check Box and Function Relationships

Check Box Option	Related Configuration Function
View Setups	Reads the [Colors] section of a configuration file which contains one or more "_RestoreViewSetup" lines. Each "_RestoreViewSetup" line defines a view setup.
System Options	BookmarkAttr ConfigSetBtnIniFilename ConfigSetLinkDBFilename ConfigSetMacroFilename ConfigSetMarkDBFilename ConfigSetSymbolDBFilename EditSetPath BufSetGlobalBackupSpec ExtCommentSearchLimit ExtDelayedColoring ExtSetUpdateDelay NameMarkFlags NameMarkThreshold KeyDelay KeyRepeat
Auto-save Options	Autosave AutosaveDir
Compiler Settings	BrowseSetFile CompilerAddBuild CompilerAssign CompilerNewExt TagSetFile CompilerAdd
Language Options	ExtColors ExtColorsAssoc ExtIndentEnable ExtIndentEnableAssoc

Table 17-2. Check Box and Function Relationships (Continued)

Version Control Setup	CheckInSetCmd CheckOutSetCmd
Filename Filters	FilterAdd FilterDeleteList
Clipboard/Scrap Options	ClipboardEnableSepStr ClipboardEnableTermStr ClipboardSetSepStr ClipboardSetTermStr ScrapSetCount

State File

Another important Hawk configuration file is the state file. The state file is stored by default in the Hawk home directory and is called `MWHAWK.PST`. `MWHAWK.PST` records the condition of Hawk at the time of exit. Storing state information is turned on by default, but can be turned off in the `Customize -> Environment -> State` dialog.

When a project is open, Window, Document and Mark information will be stored with either the project (`.pjt`) or project space (`.psp`) file, as defined on the `Customize -> Environment -> State` dialog. Other types of state information are stored in the default state file `MWHAWK.PST`. If no project is open, all information is stored in `MWHAWK.PST`.

Location of the State File

The location and name of the state file can be specified in the Hawk configuration file. If you name a location for your state file by accessing a dialog box through the menu, it will be saved in your configuration file. If your configuration file does not contain the name of your state file, and saving state information is turned on, Hawk will search for the location of the state file in much the same manner used to locate the configuration file.

The differences are as follows:

- The command line parameter for specifying the state file is `/S`.



See the section on [Command Line Parameters](#) for more information.

- The environment variable that points to the state file is `CWPST`.
- Instead of looking for `MWHAWK.INI` it will look for `MWHAWK.PST`.

If no location is dictated for the state file, and no existing file is found, the file is created in the working directory. Hawk can operate without a state file. You can turn off the

saving of state information, and Hawk will ignore any state file and the information it contains.

Similar to the configuration file, the strings that give the location of the state file may also dictate a name for the file. If the string names only a directory, the name `MWHAWK.PST` will be used. The following methods for setting the location of the state file are all valid:

- Using the environment variable:

```
set CWPST=D:\MICROWARE
```

- On the command line:

```
CW /sD:\MICROWARE\MWHAWK.PST
```

- In the `MWHAWK.INI` file:

```
StateSetFilename=H:\HOME\TOMH
```

Contents of the State File

The following are among the data kept in the state file:

Table 17-3. Data Kept in State File

Category	Types of Data in State File
Windows	Visible attributes Colors Coordinates and extents System flags Attached buffer, if any Default Window Properties for the above
Buffers	File name Output file name Tab settings Backup file specification Line and column Maximum virtual lines Current mode (hex, compact, normal) System flags Default Buffer Properties for all of the above

Table 17-3. Data Kept in State File (Continued)

Prompt Histories	Search and Replace Command Key Open file
Miscellaneous	Search flags Hawk's Window frame coordinates and extents

Command Line Parameters

Hawk supports a number of command line parameters. Command line parameters are used to specify certain conditions to apply when Hawk is started. Parameters are added to the command line by editing the file's shortcut, or through the `File Properties` selection in the `Program Manager's` menu. If you have multiple Hawk shortcuts or program items, the command line can be used to make each operate differently.

To edit the properties of a Hawk shortcut, complete the following steps:

1. Right-click the Hawk shortcut icon on your Windows desktop (or any location where the shortcut appears).
2. Select `Properties` from the pop-up menu.
3. Click the `Shortcut` tab.
4. The path and executable file are listed in the `Target` field. You can add a command line parameter at the end of this string to command Hawk to use specific settings when started.

Hawk allows the following three types of command line parameters:

- names of files to edit
- parameters or switches
- the command file, which contains command line parameters

The command line is processed before the configuration or state files have been read. This has the following effects:

- It allows the command line to specify the location of the configuration and state files.
- It allows the configuration file and state file to override the command line. Be sure to follow the stated guidelines for the use of parameters and the contents of these files. Possible conflicts are noted in the descriptions below.

Filenames

Any number of files can be specified for editing on the command line, limited only by the command line length. Wildcards can be used to specify multiple files. These files are loaded in addition to, rather than in place of, any named in the state file.

Parameters

Consider whether the configuration file should be read before or after each parameter specified. The configuration file may in some cases cause a command line parameter to be ineffective if the parameter is processed first.

There are upper case and lower case versions of each of the command line parameters, where applicable.

- Parameters specified with a lower case character will be processed before the configuration file is read.
- Parameters that use an upper case character will be processed after the configuration file is read.

Some parameters require additional information in the form of an argument. In the descriptions below, the arguments appear in italics following the parameter. When supplying an argument to a parameter, Hawk permits the argument to immediately follow the parameter, with no intervening white space, or to be separated from the parameter by white space.

The parameters listed in [Table 17-4](#) are shown preceded by a minus or dash character (-). A slash also works in most cases, however, these can be mistaken for filename paths and should be avoided.

Table 17-4. Sample Command Line Parameters

Type	Format	Description
Configuration Location Parameter	-C<configLoc> MWHAWK -c c:\source\proj1	<p>Allows you to specify the directory or file in which configuration information can be found. The <configLoc> argument names that directory or file.</p> <ul style="list-style-type: none"> • If <configLoc> names only a directory, Hawk looks for a file named MWHAWK.INI in that directory from which configuration is read. • If <configLoc> names a complete path, including filename, Hawk will attempt to read configuration information from that file. <p>For additional location information, refer to the topic Location of the Configuration File, in this chapter.</p>

Table 17-4. Sample Command Line Parameters (Continued)

Variation of Configuration Location Parameter	-C- MWHAWK -c-	Instructs Hawk not to read a configuration file.
Go to Line Number	-G <lineNumber> MWHAWK -g215	Tells Hawk to go to the line number indicated by the argument. It should follow the name of the file to which it refers, and it may be specified following each file named on the command line. This parameter will be overridden if you attempt to position the cursor in a file loaded as part of the state restoration.
Heap Allocation Parameter	-heapalloc MWHAWK -heapalloc	Instructs Hawk to use global allocation of memory. This makes it easier to detect allocation errors. Use this if you suspect your DLL has this type of memory error. <ul style="list-style-type: none"> This option is only available on the 32-bit version of Hawk.
Keymap Parameter	-K <keymap> MWHAWK -k mycua	Names a default keymap to be used by Hawk. If this parameter is used, the [DefaultKeymap] section of the Hawk configuration file (MWHAWK.INI) will not be read. The <keymap> argument to this parameter names a DLL (less the .DLL extension) containing the default keymap. The argument at the same time names the function (contained in the DLL) which initializes the keymap and makes the key assignments.
Library Parameter	-L <library> MWHAWK -Lmyutils	Designates a dynamically linked library to be loaded at startup time. This parameter is useful for testing user created or modified DLLs before a more permanent installation. It can, however, significantly increase the amount of time it takes Hawk to start up. The <library> argument is the name of the library to be loaded. If this argument does not name a path to the DLL, Hawk will look in the current directory, the Windows directory and on the PATH, in an attempt to locate the DLL.

Table 17-4. Sample Command Line Parameters (Continued)

Multi-Instance Parameter	-M MWHAWK -M	Allows you to run more than one instance of Hawk at a time. By default, multiple instances are disallowed. Your operating system must be Windows 95, Windows 98 or Windows NT for this option to work.
No Files Parameter	-N MWHAWK -N	Indicates that the state file is to be processed, except that no files from the previous session are to be restored.
No Splash Parameter	-NOSPLASH	Suppresses the display of Hawk's opening splash screen. This can save a small amount of loading time.
Paragraph Parameter	-P <section> MWHAWK -p "Windows driver"	Tells Hawk to read the named section of the configuration file. The section will be read after all of the standard portions of the configuration file—which gives it priority. The <section> argument to this parameter is the name of the section of the configuration file to be processed. If you name a section that is automatically read at startup, it will be processed twice. This section name may contain white space. If it does, however, it must be enclosed in either single or double quotes.
Limit Hawk CPU usage on multi-processor machines	-Processor = d (d is a decimal number bit mask for the allowed CPUs used by Hawk on startup)	This argument can be useful for avoiding thread related problems that appear only on multi-processor machines. It can also limit Hawk's CPU usage, allowing more for other processes. Examples: <ul style="list-style-type: none"> • '-Processor=1' limits Hawk to CPU 0. • '-Processor=2' limits Hawk to CPU 1. • '-Processor=3' limits Hawk to CPU 0 and CPU 1.

Table 17-4. Sample Command Line Parameters (Continued)

State Location Parameter	<p>-S <stateLoc></p> <p>MWHAWK -s h:\home\ericj</p>	<p>Allows you to specify the directory or file in which state information is to be found.</p> <p>The <stateLoc> argument names that directory or file. If <stateLoc> names only a directory, Hawk looks for a file named MWHAWK.PST in that directory, from which state information is read. If <stateLoc> names a complete path, including filename, Hawk will attempt to read state information from that file.</p> <p>When this parameter does not appear on the command line, Hawk looks for a state information file in a series of places, unless the saving and restoring of state information has been turned off. These places include:</p> <ul style="list-style-type: none"> • a directory or file named by the CWPST environment variable, • in MWHAWK.PST in the Working Directory, • in MWHAWK.PST in the directory in which the MWHAWK.EXE file is located, • or in a location named in MWHAWK.INI.
--------------------------	---	--

Table 17-4. Sample Command Line Parameters (Continued)

Variation of State Location Parameter	-S- MWHAWK -S-	Instructs Hawk not to read a state file. State information, however, is not automatically reinitialized, and may be used subsequently.
Execute Function Parameter	-X <function>	Names a function to be invoked as part of Hawk startup. This function should not attempt to perform any configuration, since such configuration may be overridden or reversed by the subsequent processing of the configuration and state files. The <function> argument to this parameter contains the function call. If the call contains any parameters to the function, the entire function call must be enclosed in single or double quotes. In addition, it must conform to the syntax required by LibFunctionExec.

Command Files

A command file can contain any valid command line parameters, including additional command files. Hawk identifies filenames preceded by an @ sign as command files. Command files are formatted in the following way:


```
@<commandFile>
```

Command files are useful for overcoming command line length limits and also for creating reusable, logical groupings of filenames and parameters.

White space or new lines may separate parameters within a command file. New lines are often used to promote readability.

18

Using Hawk™ Profiler



The Hawk™ Profiler profiles processes or modules on the target system. This chapter describes how to use the Hawk™ Profiler and includes the following sections:

[Setting up the Target System](#)

[The Profiler](#)

[File Menu](#)

[Options Menu](#)

[Actions Menu](#)

[Using the Profiler](#)

Setting up the Target System

Before starting the Profiler on the target, you need to have TCP/IP networking running on your target and the Profiler daemon modules loaded and running. The target modules required are `spfnppd` and `spfnppdc`. These modules are located in `MWOS\OS9000\<PROCESSOR>\CMDS` on your host system.

After you have `spfnppd` and `spfnppdc` loaded and networking is running, execute the following command:

```
spfnppd <>>>/nil &.
```



The `spfnppd` command can be added to an automated start script.

Your target is now ready to accept messages from the host and run the Profiler..




The Profiler does not support multiple connections to the target.

The Profiler

The Hawk Profiler has been re-written as a Java application; it can be installed from your Microware OS-9 product CD. To use it, you must have a Java Runtime Environment (JRE) installed on your system.

Running the Profiler

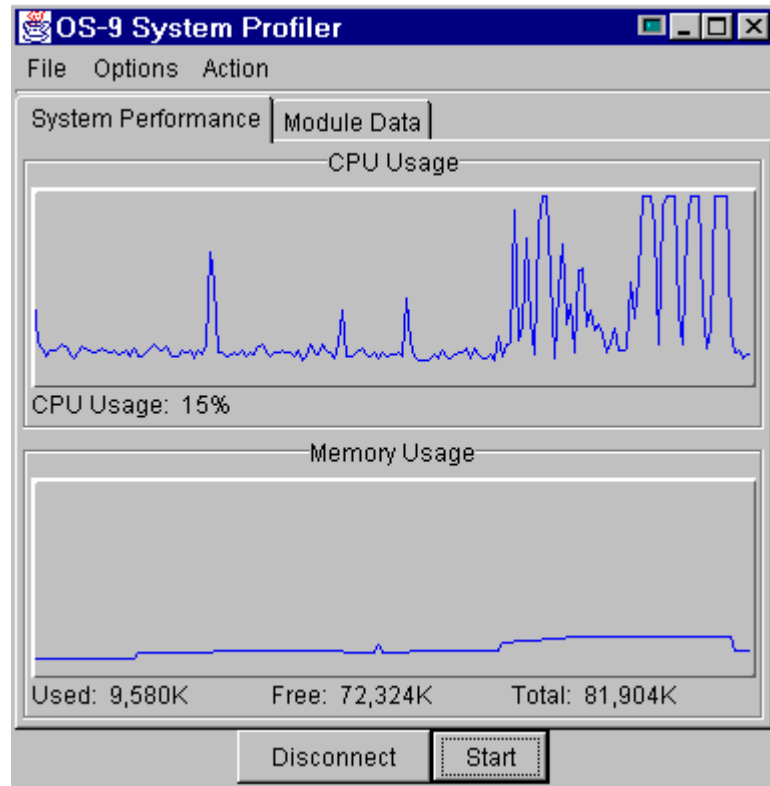
Start the Profiler, after setting up the target, by selecting **Target** -> **Profiler** or by clicking on the Profiler button  in the Hawk™ interface.

The Profiler Main Window

The main window contains the following areas:

- Title Bar
- Menu Bar
- System Performance and Module Data Tabs
- Connect and Start Buttons

Figure 18-1. Target Profiler Main Window



Title Bar

The title bar shows the name of the current file being debugged.

Menu Bar

The menu bar provides menus that you can use throughout your profiling session.

Table 18-1. Profiler Main Window Menu Entries and Actions

Entries	Actions
File	Perform operations on Profiler files. For example, start a new profiling session and save Profiler information in a file.
Actions	Configure, start, stop, and clear the Profiler.
Options	Use the Options menu to set preferences during the profile session.

Each menu is described in greater detail in the following sections.

System Performance Tab

The System Performance Tab is used as a quick reference for CPU and Memory usage for the OS-9 target. Profiling information is gathered from the target and displayed as a line chart.

Module Data Tab

This tab displays the profiling information in a more detailed manner. Each row in the table represents the profiling information for a particular module. The profiling data consists of the number of ticks charged to this module, and the percent of time that this module is executing.

To view the functions for a particular module, click on the button in the f(x) column. [Figure 18-2](#) shows the Module Data Tab.

Figure 18-2. Module Data Tab

Module	Percent	Ticks	f(x)
dhry21	87%	2,717	...
kernel	1%	42	...
spf	0%	1	...
Idle	10%	339	
Total		3,099	

- Function breakdowns are only available if a `.stb` module for that module can be found on the host.

Start/Stop Button

Use the `start` button to begin profiling the target system. The button toggles between stop and start, depending on the current state of the Profiler.

Connect/Disconnect Button

Use the `Connect` button to establish communication with the OS-9 target system. This will toggle between `Connect` and `Disconnect` depending on the current state of the Profile.

File Menu

Use the File menu to perform operations on Profiler files.

Table 18-2. Profiler File Menu Entries and Actions

Entry	Action
Save	Save the current Profiler information to the current Profiler file.
Save as text	Save information as text.
Restore	Open a file that has been previously saved and display the information in the Profiler windows.
Exit	Exit the Profiler.

Options Menu

Use the Options menu to set preferences during the profile session.

Table 18-3. Profiler Options Menu

Entry	Action
Symbol Paths	This shows a dialog window which allows pathnames to be selected where OS-9 Symbol Files (.stb) can be located.
Sample Rate	The update interval of the profiling information can be controlled by this dialog.
Show All Modules	Normally, only modules that have executed will show up in the Module Data table. However, all modules on the OS-9 target system can be viewed by selecting this option.

Actions Menu

Use the Actions menu to control the way profiling information is obtained.

Table 18-4. Profiler Actions Menu Entries and Actions

Entry	Action
Start	Start profiling and tell the target daemon to start sending information. You can also click Start (located at the bottom of the window) or press Ctrl-c to start profiling.
Stop	Stop profiling. You can also click Stop (located at the bottom of the window) or press Ctrl-t to stop profiling.
Clear	Clear the current timing information to zero.
Connect	Connect to target.
Disconnect	Disconnect from target.

Using the Profiler

This section describes some of the more commonly used Profiler functions. Information gathered from the profiler can be used in the following two ways:

- As a quick reference to see how the OS-9 target is performing.
- As a tool to investigate performance during a specific task.
- It may be useful to increase the update interval to be greater than the specific task time. Also, increasing the tick rate of the OS-9 target to 1000 can increase the resolution of the information.

Viewing Functions

The functions of a module can be viewed by clicking on the button in the f(x) column. This will bring up a separate window containing all the functions that have executed. If an OS-9 Symbol file can not be located, a file selection dialog box will appear allowing you to find the symbol file.

Saving the Profile Data

The profiling information can be saved to a text file by selecting the **File->Save...** menu option.

Changing the Order in which Modules/Functions Appear

By default, when you return to the main window of the Target Profiler, the modules that you have selected appear in the order you entered them in the Modules to Monitor window. You can also display the modules in any of these orders:

- Alphabetical Order.
- CPU Usage Order is relevant due to the fact that after you have begun collecting profiling information, the module using the most CPU time will be at the top of the list, and the module using the least CPU time will be at the bottom of the list.

To change the order, click on the table heading for that column.

Profiling Selected Modules

To profile your selected modules, select **Start** from the Actions menu. For each module selected, the Target Profiler displays the following information in the timing analysis window:

- The number of tick boundaries on which this module was running.
- The percentage of the time that the module has spent in the CPU.
- A bar graph showing the amount of time the process has spent in the CPU.
- The daemon `spfnppdc` must be loaded into the Target memory.

Changing the Update Interval

By default, the timing analysis window is refreshed every five seconds.

The following steps guide you through changing the update interval for refreshing the timing analysis window.

1. Select **Sample Rate...** from the Options menu.
2. Enter the update interval (in seconds) and click **OK**.

Stopping Profiling

When you are ready to stop profiling, select **Stop** from the Actions menu or click **Stop** at the bottom of the Profiler window. The information from the profiling session remains on your screen.

Clearing the Profiling Data

You can clear the data that you have collected from your profiling session by selecting **Clear Data** from the Actions menu. The module names that you profiled are retained, but the remaining information is cleared to zero.

Exiting the Target System Profiler

To exit the Target System Profiler, select **Exit** from the File menu.

19

Using HawkEye™

The HawkEye™ software application is a GUI-based execution visualization tool for the OS-9 operating system.

You can use the HawkEye visualization tool to capture and analyze a log of events during the execution of a program on an OS-9 target machine.

Examples of events include:

- System calls such as process forks, signals, and exits
- Context switches
- Interrupts



HawkEye is documented in the *Using HawkEye* manual.

Starting HawkEye

You can start HawkEye from the main Hawk user interface by selecting **Target -> HawkEye . . .** from the main toolbar.



TagsWNN Utility



`TAGSWnn` (`TAGSW16` or `TAGSW32`, depending on your platform) is the Tags program supplied with Hawk to automatically generate a tags database and compile it into a format that can be used by the built-in browser. The program is called when you select `Build Tags` from the `Project` menu. You may not ever need to run the program from the command line, but in the event that you do, or if you just want to know more about the program, its options are briefly described here.

`TAGSWnn` is based on the GNU Tags program written by John Kerchival. The primary changes made to it are to allow output in the Starbase Compiled Tags format. This is the `-p` option, which is the only option we have added. Other options are as described in the `TAGS.DOC` file supplied with the GNU Tags program and provided with Hawk.

Usage

Example

```
TAGSWnn { [OPTIONS] [SOURCEFILE|@LISTFILE] }
```

TagsWnn Command Line Options

An option summary follows:

`-h, -?`

Obtain a detailed help screen directed to standard output.

`@LISTFILE`

Use `LISTFILE` as a "response" file. This file lists input filenames (with or without wildcards) one at a time. Filenames may be separated by `+',',',',';'` or by whitespace. In addition, comments are allowed within the `LISTFILE`. Comments are delimited by placing a pound sign `#` before the comment. This is very similar to comments allowed in a makefile except that comments are allowed on any line or at the end of any line, start at the `#` and go to the end of the current line. There must be at least one character between the filename and the comment character (`+',',',',';'` or whitespace) to differentiate between the beginning of comment and a filename character (since `#` is a valid element of a filename).

`-x{EXCLUDEFILE|@LISTFILE}`

Excludes the files specified by `EXCLUDEFILE` or



excludes all files listed in `LISTFILE` using the same syntax described above.

`-tTAGFILE`

Add new generated tags to `TAGFILE`. This file may or may not exist. All tags from `TAGFILE` which were derived from files currently being parsed will be removed during the merge phase. This tagfile is assumed to be in one of this utilities output formats. If sorting is specified then new tags will be merged in correct order with current case sensitivity, otherwise tags will be placed at the beginning of the new resulting tag file (this will result in quicker responses during tag searches while editing). If `-m` or `-s` are used this switch is ignored (all output is to `stdout`). The behavior regarding existing files is determined by the case of the switch as follows:

`-t` (lower case) creates and outputs to a file overwriting any currently existing file.

`-T` (upper case) merges all output to the tagfile if there is an already existing file.

`-pCOMPILEFILE`

Convert output tag file (specified by `-t`) into a compiled file suitable for use with Hawk. The output flag (`-t`) and the Hawk output format flags (`-oc`) must be specified for this flag; otherwise, it is ignored.

`-lLOGFILE`

Output all activity to `LOGFILE`. The log file will be created in a `LISTFILE` format (so it is suitable as input using the `@LISTFILE` syntax). The behavior regarding existing files is determined by the case of the switch as follows:

`-l` (lower case) creates and outputs to a file overwriting any currently existing file.

`-L` (upper case) appends all output to the logfile if there is an already existing file.

`-o[options]`

This switch is used to determine the output format to the output stream. `[options]` may be one of the following:

```
e  Epsilon (>= V6.0) tag format
(  tokenString {tab} fileName {tab} characterOffset
   {tab} line )
```

This format is used by the Epsilon editor (V6.x) created by Lugaru Software and specifies the token identifier, the file name (including full path,

normally), the character offset of the beginning character (starting at character 0) and the line which that offset is located on.

Epsilon (<= V5.03) tag format
(tokenString;fileName;characterOffset)

This format is used by the Epsilon editor (V4.x and V5.x) created by Lugaru Software and specifies the token identifier, the file name (including full path, normally) and the character offset of the beginning character (starting at character 0).

g GNU tag format(tokenString {tab} fileName
{tab} /\$line^)

This format is used by GNU's EMACS editor, originally written by Richard Stallman and widely used in the UNIX community. This is also the format created by its companion utility "ctags" which does very simple function header tagging.

s Space-Delimited format(tokenString
fileName lineNumber)

This format is the simplest format available and requires very little parsing and is very simple to import into foreign formats (such as database formats).

m Microsoft Error format(tokenString
fileName(lineNumber))

This format has an advantage in that it has been around for quite some time and a fair amount of effort has been expended to parse this format and move to the location in the source specified during compilation stages. Many macros may be modified to use this type of tag format with very minor changes.

c Hawk tag format(tokenString
FileName(lineNumber) tokenType)

This format is a minor variant of the Microsoft format but is generally used in conjunction with a compiled database file format (see -p above). This is the format used by Starbase's Hawk editor.

-a [options]

This switch is used to specify the types of tokens for which tags are generated for tagging of assembly files. All token types are tagged as the default (-afd1msu). Source modules are expected in 80x86 assembly using MASM/TASM syntax. The location of the -a switch on the command line is



	important. All files (and files found in a LISTFILE) will be tagged using assembly tagging (and the options specified on that switch) until another -a or -c switch is found. Order is not important for the options to this switch.
f procedure labels	(token proc)(proc token) This is a mnemonic for function (which has nothing to do with a procedure call in assembly. This option specifies tagging of the "proc" keyword.
d definition labels	(token equ const)(token db declaration) This option specifies tagging of defines and definition labels such as the tokens "equ", "db", "dq", "dw", "df".
l local labels	(token label)(label token)(token:) This option specifies tagging of local labels (labels of local file duration). This includes the keyword "label" as well as the shorter ':' notation.
m macro labels	(token macro)(macro token) This option specifies tagging of defined macros using the keyword "macro".
s struc labels	(token struc)(struc token) This option specifies tagging of structure definitions defined using the keyword "struc".
u union labels	(token union)(union token) This option specifies tagging of union definitions defined using the keyword "union".
-c [options]	This switch is used to detail the token types to tag in C and C++ source files. All token types are tagged by default (-cdmstekuvcfpxi). Source files are expected in standard ANSI 2.0 C/C++ syntax. The location of the -c switch on the command line is important. All files (and files found in a LISTFILE) will be tagged using C tagging (and the options specified on that switch) until another -a or -c switch is found. Order is not important for the options to this switch.
d defines	(#define token statement) This option specifies that defines are to be tagged (preprocessor defines). This does not include macros which are an extended use of the #define preprocessor directive.

m	macro labels	(<code>#define token() statement</code>) This option specifies tagging of macros defined via use of the preprocessor <code>#define</code> directive.
s	struct globals	(<code>struct token {}</code>) This option specifies tagging of structures defined via use of the <code>struct</code> keyword and implicitly defined within C++ syntax variations.
t	typedef globals	(<code>typedef declaration token, token, ...</code>) This option specifies tagging of identifiers defined via use of the <code>typedef</code> keyword.
e	enum globals	(<code>enum token {}</code>) This option specifies tagging of enumerations defined by use of the <code>enum</code> keyword.
k	enum konstants	(<code>enum { token, token, token, ...}</code>) Note the spelling of constants with a 'k'. This option specifies tagging of enumeration constants within declared enumerations.
u	union globals	(<code>union token {}</code>) This option specifies tagging of unions defined via use of the <code>union</code> keyword.
v	global variable	(<code>declaration token, token = {}, token, ...</code>) This option specifies tagging of global variable declarations.
c	global class	(<code>class token: {}</code>) This option specifies tagging of class definitions specified via use of the <code>class</code> keyword.
f	function definitions	(<code>token() declaration {}</code>) This option specifies tagging of function declarations.
p	prototypes	(<code>token();</code>) This option specifies tagging of prototypes.
x	extern defines	(<code>extern declaration</code>) (<code>extern "C" declaration</code>) (<code>extern "C" { declaration; declaration; ... }</code>) This option will specify that tags which have the extern storage class are to be output. The <code>x</code> option is a modifier and will only be effective when other options are used (for example, <code>-cpx</code> must be specified to obtain extern prototypes, <code>-cx</code> alone

	yields nothing). Note also that the <code>-cx</code> modifier has no effect for function, define and macro tags which are tagged only according only to the <code>x</code> , <code>d</code> and <code>m</code> options, respectively. This modifier may be placed anywhere within the options list.
<code>i</code>	<code>static declarations</code> (static declaration) This option will specify that tags which have internal static storage class are to be output. The <code>i</code> option is a modifier and will only be effective when other options are used (for example, <code>-cvi</code> must be specified to obtain static variable declarations, <code>-ci</code> alone yields nothing). Note also that the <code>-ci</code> modifier has no effect for define and macro tags which are tagged only according only to the <code>d</code> and <code>m</code> options, respectively. This modifier may be placed anywhere within the options list.
<code>-d</code>	This flag specifies that all input listfiles and exclude response files should be deleted once parsed. This allows an automated list file generation which is cleaned up by the tags package. See description of <code>LISTFILE</code> below.
<code>-j</code>	This is the junk filter switch to allow the filtering of functions and declarations which are overloaded operators in C++. For example, if the junk filters are enabled then the declaration <code>"inline myType operator+(MyType m1, MyType m2);"</code> would not be tagged for <code>"+"</code> which is normally a standard C delimiter token and operator. The junk filter if enabled will filter all standard C delimiters from the output.
<code>-q</code>	This is the quiet switch and will suppress normal status output to <code>stderr</code> and program version information.
<code>-r</code>	This switch will suppress the default output of the full file path name and will specify the use of relative pathnames in the generated output.
<code>-n</code>	This switch will suppress sorting of the tag output. (This is often used in conjunction with GNU or Epsilon style tags.)
<code>-i</code>	This switch specifies the use of a case-sensitive sort. (Normally the <code>-i</code> option is used for a case-insensitive sort is used).





Command I/O



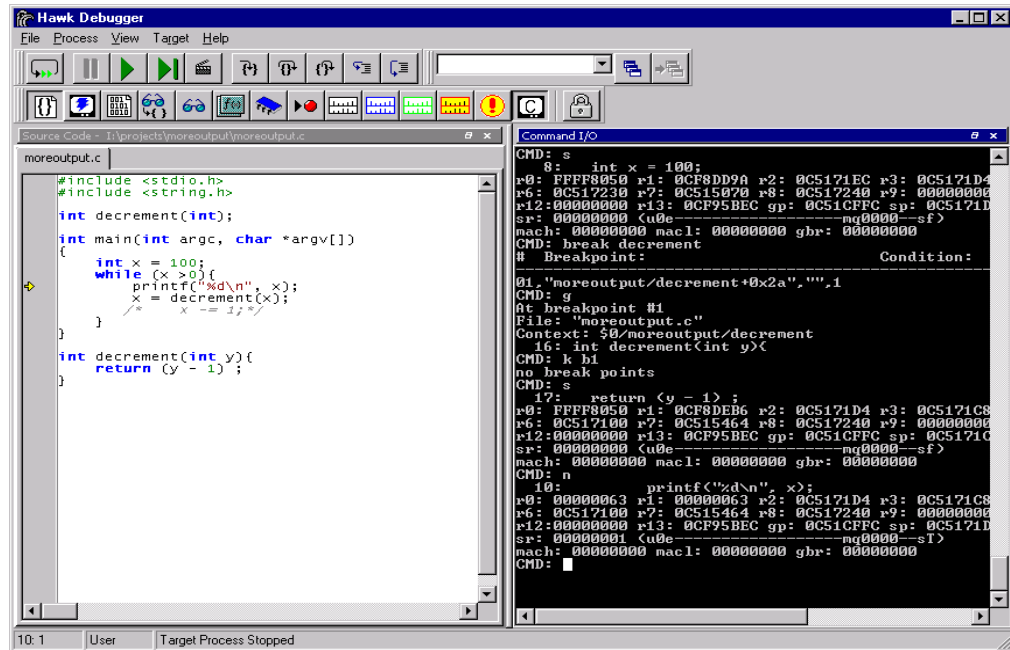
This appendix explains the commands that can be entered into the Command I/O area of the main Hawk Debugger window.



Commands

The View window pull-down menu has, as one of its entries, [Command I/O](#). Toggling this option on or off displays or hides the command I/O area in the Debugger main window (shown in [Figure 19-1](#)).

Figure 19-1. Hawk Debugger Window



The following pages list, in alphabetical order, the Debugger commands you can enter in this area. The characters shown in brackets specify the alternate form of the command. For example, the assignment command can be entered as "a" or "assign".

asm, .

Syntax:

asm

Display the current register values and current machine instruction. A period (.) has the same effect as asm.

a[ssign]

Syntax:

a[ssign] <C_expr> = <C_expr>

Set the value of a program object.

[See Also:](#) p[rint]

at[tach]

Syntax:

```
at [tach] <target>@0
at [tach] <target>@<procnum>
at [tach] <module>
at [tach]@SEARCH <start> <end>
```

Attach to a target system, process, module, or modules in a given range.

b[reak]

Syntax:

```
b[reak] [<location_expr>] [@wh[en] <C_expr>] [@co[unt]
<num>]
```

Set a breakpoint at a specified line number, result of a `<C_expr>`, or upon a when `<C_expr>` becoming true.

[See Also](#): `go`, `gostop`, `kill`, `next`, `print`, `return`

con[text]

Syntax:

```
con[text] [<scope_expr>]
```

Display the complete scope expression of an object.

dil[ist]

Syntax:

```
dil[ist] [<location_expr>] [@ [<count>]]
```

Display the current C source line and the assembly code which maps to the current C source line starting at the address specified by `<location_expr>`.

di[sasm]

Syntax:

```
di[sasm] [<C_expr>] [@ [<count>]]
```

Disassemble and display memory starting at the address specified by `<C_expr>`.

- If you specify `<count>`, `<count>` machine lines are disassembled and displayed.
- If you do not specify `<count>`, sixteen machine lines are disassembled and displayed.



download

Syntax:

```
download <file> <address expression>
dl <file> <address expression>
```

Download the contents of the specified file into the target system's memory at the specified address.

d[ump]

Syntax:

```
d[ump] [ [<C_expr>] [@ [<count>]
```

Return a formatted display of the physical contents starting at the address specified by <C_expr>.

- If you specify <count>, <count> lines of information are displayed.
- If you do not specify <count>, sixteen lines of information are displayed.

fi[nd]

Syntax:

```
fi [nd] [<name>]
```

Display all scope expressions found for <name>.

[See Also:](#) wfi [nd]

fo[rk]

Syntax:

```
fo[rk] [<machine>@]<program> {<program arg>}
["<path redirection>"]
```

Begin executing the specified program to begin a debugging session.

If no fo[rk] arguments are specified, fo[rk] uses the last command arguments used in fo[rk].

f[rame]

Syntax:

```
f[ame] [[±] <number>]
```

Display stack frame information.

If no arguments are specified, the name of each calling function, the location from which it was called, and the frame number are displayed.

If f[ame] is followed by <number>, the stack frame context is changed to <number>.

g[o]

Syntax:

```
g[o] [<location_expr>]
```

Execute the program starting at the current location. If `<location_expr>` is specified, the program runs until `<location_expr>` is reached.

[See Also](#): `break`, `gostop`, `trace`, `wait`, `watch`

gostop or gs

Syntax:

```
gostop [<number>]
```

```
gs [<number>]
```

Execute the specified number of machine instructions in the current subroutine. This command is identical to the `trace` command, but does not trace through subroutines.

[See Also](#): `break`, `go`, `trace`, `wait`, `watch`

i[nfo]

Syntax:

```
i[nfo] [<scope>]
```

Return information about specified program objects and the current program location.

If no argument is specified, `i[nfo]` returns the current location of the program.

k[ill]

Syntax:

```
k[ill] [<breakpoint>] { [,<breakpoint>]
```

Remove all specified breakpoints. `k[ill]` uses the following notation for breakpoints: `b1` = breakpoint #1, `b2` = breakpoint #2, and so on. `k[ill]` can use either this notation for breakpoints or a location expression similar to `break`.

[See Also](#): `break`



l[ist]

Syntax:

```
l[ist] [<list arg> [,<list arg>]]
```

Return a source listing of the specified file or portion of the file. A list argument may be a file name, line number, or a scope expression resulting in the following:

- A block number.
- A function.
- A line number.

If a beginning line number is specified, you can use a second line number to specify the ending line number (for example, `list 1 10` will list lines 1 through 10 of the current source file).

lo[cal]

Syntax:

```
lo[cal]
```

Display the names and values of all local variables.

l[o]g

Syntax:

```
l[o]g <pathlist>  
l[o]g @ off
```

Write Command I/O window commands to `<pathlist>`. `<pathlist>` is relative to your current directory. Enter `l[o]g @ off` to close the log file.

mc[opy]

Syntax:

```
mc[opy] <dest> @ <source> @ <length>
```

Copy `<length>` bytes of memory from `<source>` to `<dest>`. All arguments are C expressions.

mf[ill]

Syntax:

```
mf[ill] [n] <begin> @ <end> @ <value>
```

Fill bytes of memory in the address range from `<begin>` to `<end>` with `<value>`.

All arguments are C expressions.

An optional 'n' character can appear after the command for compatibility with `mfl` and `mfw`, but otherwise has no effect.

[See Also:](#) `mfilllong`, `mfillword`

`mfilllong`

Syntax:

```
mfilllong [n] <begin> @ <end> @ <value>
mfl [n] <begin> @ <end> @ <value>
```

Fill longwords of memory in the address range from `<begin>` to `<end>` with `<value>`.

All arguments are C expressions.

An optional 'n' character can appear after the command to suppress longword boundary checking. That is, if `mfln` is used, any partial longwords at the end address will be filled with a portion of the `<value>`. If plain `mfl` is used, the filling will stop at the last complete longword at the `<end>` address.

[See Also:](#) `mf[ill]`, `mfillword`

`mfillword`

Syntax:

```
mfillword [n] <begin> @ <end> @ <value>
mfw [n] <begin> @ <end> @ <value>
```

Fill words of memory in the address range from `<begin>` to `<end>` with `<value>`. All arguments are C expressions.

An optional 'n' character can appear after the command to suppress word boundary checking. That is, if `mfwn` is used, any partial words at the end address will be filled with a portion of the `<value>`. If plain `mfw` is used, the filling will stop at the last complete word at the `<end>` address.

[See Also:](#) `mf[ill]`, `mfilllong`

`m[onitor]`

Syntax:

```
m[onitor] <vector>
m[onitor] <start>-<end>
m[onitor] <vector> [@DISPLAY] [@USER] [@SYSTEM]
m[onitor]
```

Monitor a vector, monitor a range of vectors, or display vectors being monitored. This command is only available if the debugger is being used as a system debuffer (system-state debugging configuration).

`ms[earch]`

Syntax:



```
ms [earch] [n] <begin> @ <end> @ <value> [@ <mask>]
```

Search bytes of memory from <begin> to <end> for <value> and display each line where <value> is found.

All arguments are C expressions.

An optional 'n' character can appear after the command for compatibility with `msl` and `msw`, but otherwise has no effect.

[See Also](#): `msearchlong`, `msearchword`

msearchlong

Syntax:

```
msearchlong [n] <begin> @ <end> @ <value> [@ <mask>]
```

```
msl [n] <begin> @ <end> @ <value> [@ <mask>]
```

Search longwords of memory from <begin> to <end> for <value> and display each line where <value> is found. All arguments are C expressions.

An optional 'n' character can appear after the command to suppress longword boundary checking. That is, if `msln` is used, any partial longwords at the end address will be searched for a portion of the <value>. If plain `msl` is used, the searching will stop at the last complete longword at the <end> address.

[See Also](#): `msearch`, `msearchword`

msearchword

Syntax:

```
msearchword [n] <begin> @ <end> @ <value> [@ <mask>]
```

```
msw [n] <begin> @ <end> @ <value> [@ <mask>]
```

Search words of memory from <begin> to <end> for <value> and display each line where <value> is found.

All arguments are C expressions.

An optional 'n' character can appear after the command to suppress word boundary checking. That is, if `mswn` is used, any partial words at the end address will be searched for a portion of the <value>. If plain `msw` is used, the searching will stop at the last complete word at the <end> address.

[See Also](#): `msearch`, `msearchlong`

n[ext]

Syntax:

```
n [ext] [<number>]
```

Execute the specified number of executable statements of the program. If you do not specify a <number>, one statement is executed.

`n [ext]` executes functions as a single statement.

[See Also](#): `go`, `step`

`ow`

Syntax:

```
ow {<C_expr>}
```

Set a watch point at the address specified by `<C_expr>`. The address is watched for any access: read, write, or execute. Watch points are created in the “active” state and will trigger the next time they are hit. When a watch point is encountered, you have the option of deactivating all of your watch points. If you do not deactivate them, it may be impossible to advance the target since it’s likely the watch point will be hit repeatedly.

If no `<C_expr>` is specified, the current list of watch points is shown. Use this list to get the watch point numbers for use with `owk`, `owa`, and `owd`.

Note

This command is only available when Hawk is being used as the system debugger (system-state debugging configuration).

The availability of this command is highly target dependent. Not all targets have this facility implemented.

The number of watch points available any any given time is also target dependent.

Whether a watch point is triggered on read, write, and/or execute depends on the target’s ability to set watch points based on those actions. For example, a target may only be able to trigger a watch point on read or write.

[See Also](#): `owa`, `owd`, `owk`

`owa`

Syntax:

```
owa <num>
```

Activate the watch point specified by `<num>`. When the target encounters an activated watch point control is returned to the debugger. If the specified watch point is already active, it remains active. Use the `ow` command to determine the status of your watch points.

`owd`

Syntax:

```
owd <num>
```



Deactivate the watch point specified by `<num>`. Deactivated watch points are deleted from the target. If the specified watch point is already inactive, it remains inactive. Use the `ow` command to determine the status of your watch points.

owk

Syntax:

```
owk <num>
```

Kill the watch point specified by `<num>`. The watch point specified by `<num>` is deactivated and deleted from the list of available watch points. Use the `ow` command to determine the status of your watch points.

p[rint]

Syntax:

```
p[rint] <C_expr>
```

Print the value of the specified C expression.

pr[ocs]

Syntax:

```
pr[ocs] [$<process number>]
```

Display process information or change the current default process.

If specified, `<process number>` indicates the process that is to become the current default process.

q[uit]

Syntax:

```
q[uit]
```

Terminate the debugging session or abort an interrupted function call.

re[ad]

Syntax:

```
re[ad] [<pathlist>]
```

Read Command I/O commands from `<pathlist>`. The `<pathlist>` is relative to your current data directory. The `l[o]g` command may be used to create the file referred to in `<pathlist>`.

[See Also:](#) `log`, `step`

rel[ease]

Syntax:

```
rel[ease] <vector>
```

```
rel [ease] <start>-<end>
```

```
rel [ease]
```

Release the specified vector or vectors, or release all vectors. This command can only be used when Hawk is being used as a system debugger (system-state debugging configuration).

reset

Syntax:

```
reset
```

```
rst
```

Reset the target system. This command can only be used when Hawk is being used as a system debugger (system-state debugging configuration).

r[eturn]

Syntax:

```
r [eturn] [<number>]
```

Execute the program until the current function returns to the calling function or a breakpoint is encountered.

If you specify <number>, `r [eturn]` executes until it returns to the specified number of callers above the current stack frame.

[See Also:](#) `frame`

se[tenv]

Syntax:

```
se [tenv] <environment_name> <environment_definition>
```

Set a shell-type environment variable for use by the debugger's child processes.

<environment_name> and <environment_definition> are strings that are stored in the environment list by the current shell.

[See Also:](#) `unsetenv`

s[tep]

Syntax:

```
s [tep] [<number>]
```

Execute the specified number of executable statements of the program. If you do not specify <number>, one statement is executed.

[See Also:](#) `go`, `next`



sy[mbol]

Syntax:

```
sy [mbol] [<C_expr>]
```

Display the result of the expression as a symbolic expression.

t[race]

Syntax:

```
t [race] [<number>]
```

Execute the specified number of machine instructions. If `<number>` is not specified, `t [race]` executes a single instruction.

Trace is identical to `gostop`, but traces through subroutines as well.

[See Also:](#) `gostop`

unse[te]nv]

Syntax:

```
unse [te]nv]
```

Delete an environment variable from the environment list.

[See Also:](#) `setenv`

wait

Syntax:

```
wait [${<process number>}]
```

Monitor a process until it suspends execution and then print the current location for that process.

`<process number>` specifies the process to suspend. If you do not specify `<process number>`, `wait` monitors the current default process.

[See Also:](#) `asm`

wfi[nd]

Syntax:

```
wfi [nd] <wildcard pattern>
```

Locate all symbol names that match a specified wildcard pattern. The wildcard pattern can contain `*` to match any sequence of characters, `?` to match any single character, and `[{-}<x>-<y><z>]` to match any single character in the range `<x>` to `<y>` or the character `<z>`. If the first character of the set is `'-'` it will match a character not in the set. The symbols are compared without consideration for uppercase or lowercase.

x{string}

Syntax:

```
x{string}
```

Execute an “OEM extension” command and display the result. This command attempts to execute {string} as a RomBug OEM extension command. The list of available OEM commands varies from target to target, but there is usually some help text displayed by issuing the `x?` command.

Note

This command is only available when Hawk is being used as the system debugger (system-state debugging configuration).

The RadiSys OEM extensions are in the module called `oemcmd`. This module must be initialized prior to attempting any `x` commands. The `init` module can be modified to initialize this module via the configuration wizard or the `oemcmd` module can be dynamically initialized with the `p2init` utility.

