

CodeWright 6.5

User Guide

code1
Vw)

Update

PROGRAMMING

TEAM
objects

CLASSES

JAVA
PYTHON

PERL

C++
HTML

VB

XML

edit
codesense
METHODS

```
String rs;  
  
cellColor = Color  
inputColor = new  
inputFont = new  
titleFont = new  
title = getParam  
if (title == null)  
    title = "Spz  
{  
rs = getParamete  
if (rs == null)  
    rows = 10;  
else {  
    rows = rs  
    rows = Integer  
}  
rs = getParamete  
if (rs == null)  
    columns = 5;  
} else {  
    columns = In  
}  
cells = new Cell  
char i[] = new c  
for (int i=0; i  
    for (int j=0  
        cells[i][j] = ne  
            Color Lig  
            Color b1c  
            cellColo  
            cellLig  
            cellMed  
            1/10) = (char) ((  
rs = getParamete  
if (rs != null)  
    {  
        Dimension d = s;  
        inputArea = new  
        inputColor, Col  
        resize(columns *  
            (rows +  
            }  
        public void  
        if (selectedRow  
        return;  
        cells[selectedRc  
        repaint();  
        public void  
        stop = 0;  
        }  
        public void  
        isStopped = false;  
        public void  
        for (int i=0; i  
        for (int j=0; j  
        if (cells[i][j]  
        )  
        )  
        public void  
        if (selectedRow  
        return;  
        cells[selectedRc  
        int();  
        HTML  
        public void  
        if (fullUpdate  
        int cx, cy;  
        g.setFont(t;  
        for (int i=0  
        for (int j=0; j  
        if (cells[i]  
        cx = (j * cellW  
        cy = ((i+1) * c  
        cells[i][j].pair  
        )  
        )  
        )  
        } else {  
        paint(g);  
        fullUpdate =
```

Copyright © 1991 – 2000 Starbase Corporation. All rights reserved. Premia Corporation is now owned by Starbase Corporation.

The following copyright message is required due to the inclusion of CTL3D.DLL with our product:

Portions © Microsoft Corporation, 1985-1995. All rights reserved.

Publication History

October, 1991	First Release
February, 1992	Updated for 1.1
June, 1992	Updated for 2.0
August, 1993	Updated for 3.0
August, 1994	Updated for 3.1
November, 1995	Reformatted and updated for 4.0
January, 1997	Updated for 5.0
May, 1999	Reformatted and updated for 6.0
October, 1999	Updated for 6.0b
April, 2000	Updated for 6.0d
Aug, 2000	Updated for 6.5

Trademarks

Starbase® and CodeWright® are registered trademarks of Starbase Corporation.

CodeSense™, CodeFolio™, ChromaCode™ and ChromaCoding™ are trademarks of Starbase Corporation.

Code Composer Studio™ and Texas Instruments™ are trademarks of Texas Instruments Incorporated. TICCSync only works with Code Composer Studio™ from Texas Instruments™. MERANT™ is a trademark of MERANT International Limited. PVCS® is a registered trademark of MERANT Solutions Incorporated.

Borland C++®, Borland C++ Builder®, BRIEF®, and Delphi® are registered trademarks of Borland.com, a division of Inprise Corporation.

Microsoft® is a registered trademark of Microsoft Corporation.

Windows™, Visual C++™, and Visual Basic™ are trademarks of Microsoft Corporation.

Epsilon™ is a trademark of Lugaru, Inc.

AppBasic uses WinWrap® Basic, Copyright © 1993-1996, Polar Engineering and Consulting.

Sentry Spelling-Checker Engine, Copyright © 1993-1997, Wintertree Software Inc.

Contact Information

Starbase Corporation
9615 SW Allen Blvd., Suite 107
Beaverton, Oregon 97005
Phone: (503) 641-6000; Fax: (503) 641-6001
Email: codesales@starbase.com or codesupport@starbase.com
World Wide Web: <http://www.starbase.com>

Table of Contents

Publication History	ii
Trademarks	ii
Contact Information	ii

TABLE OF CONTENTS	III
--------------------------	------------

CHAPTER OUTLINE	1
------------------------	----------

Chapter 1: Introduction	1
Chapter 2: Run CodeWright for the First Time	1
Chapter 3: Build Your First Project	1
Chapter 4: Command Key, Libraries, & Environment	1
Chapter 5: View Setups and Language Support	1
Chapter 6: Editing & Printing	2
Chapter 7: Projects, Project Spaces, and Workspaces	2
Chapter 8: Set up a Compiler	2
Chapter 9: Version Control	2
Chapter 10: Synchronization	3
Chapter 11: Search and Replace and Navigational Tools	3
Chapter 12: Checking and Reformatting Files	3
Chapter 13: Custom Interface	3
Chapter 14: File Loading, Backup and FTP	4
Chapter 15: Large Files	4
Chapter 16: Extend CodeWright	4
Chapter 17: UNIX	4
Chapter 18: Configuration Files & Command Line Parameters	4
Appendix A: TagWnn Utility	5

1- INTRODUCTION	7
------------------------	----------

What's In This Chapter	7
What Makes CodeWright Different	7
Additional Product Support	8
Web Page	8
Internet Mail	8
Fax	8
Phone Support	8
Key Editor Features	9

2- RUN CODEWRIGHT FOR THE FIRST TIME	17
---	-----------

Configuration Wizards	17
A First Look	18
The Menu Bar	19

File Menu	19
Edit Menu	20
Search Menu	20
Project Menu	21
Text Menu	22
Document Menu	22
Customize Menu	23
Tools Menu	25
Window Menu	26
The Difference between Windows and Documents	26
Details of the Window Menu	26
Help Menu	27
Output and Project Windows	27
Output Window	27
The Project Window	28
More on the Open Tab	29
The Standard Toolbar	30
3- BUILD YOUR FIRST PROJECT	33
Making CodeWright Projects and Project Spaces	33
Creating a Project Space	33
Setting Project Defaults	34
Creating a Project	34
Adding Files to a Project	35
Project Properties	36
Project Tools	36
Project Window File View	36
4- COMMAND KEY, LIBRARIES, & ENVIRONMENT	39
API Command Dialog/Prompt	39
API Command Key	40
Command Completion	40
API Command Completion Example	41
Customize Libraries: Loading CodeWright Add-Ons	42
General Environment Settings	42
5- VIEW SETUPS AND LANGUAGE SUPPORT	45
View Setups: Colors, Window Attributes, Scrollbars, Fonts, Etc.	45
CodeWright View Setups	47
Default View Setup	48
Output Window View Setup	48
Using View Setups	48
CodeWright Language Support	49
Language Support Lexers and DLL's	49

Language DLLs	50
Customize Libraries	50
ChromaCoding Lexers	51
Creating a Lexer	52
Configuring Options in the Language Dialog	60
File Type List	61
Options Tab	61
Tabs/Indenting Tab	62
Templates Tab	62
Coloring Tab	62
CodeSense Tab	64
Format Tab	64
Comments Tab	64
Adding a New File Type to the Language Dialog	64
Aliasing	65
Creating a Language Support DLL	66
6- EDITING & PRINTING	67
Templates and Brace Expansion	67
Templates	67
Language Specific Templates	67
Creating and Modifying Language Specific Templates	68
Non-Language-Specific Templates, Function and File Headers, and	
Macros in Templates	69
Template Macros	70
CodeFolio Snippets	74
Using an Existing Code Snippet	75
Adding a Code Snippet	77
Creating a Snippet from the Current Document or Clipboard	77
Deleting or Renaming a Snippet	78
Editing a Snippet	78
Adding or Removing Snippets Directories	79
Brace Matching and Brace Expansion	80
Brace Function: Finding Unmatched Braces	80
Brace Highlighting	81
Brace Locating	81
BraceFindEx	81
Brace Expansion	82
Align Beginning and End of Block	82
Indenting	83
Setting Spaces and Tabs	83
Seek Indentation and Smart Indenting	84
Block Alignment	84

Name Completion	85
CodeSense	86
Where CodeSense Gets its Information	86
Library and Project Databases	86
CodeSense for Files that are Open in CodeWright	88
CodeSense Global Configuration Dialog	88
Create CodeSense Library Database	89
Edit CodeSense Library Database Location	90
Delete CodeSense Library Database	90
Parser Priority/Resource Use	91
CodeSense Databases	91
Database Files	92
Database Corruption	93
CodeSense: Main Features	94
Name Completion	94
Auto-list Members	95
Auto-type Info	95
Auto-Parameter Info	96
Extending CodeSense Functionality	96
ANSI and Unicode CodeSense Translations	96
Disable CodeSense for Sections of Code Only	96
Project Matches in Non-Project Files	97
Consolidate Matching Lookup Definitions	97
Symbol Lookups	97
Troubleshooting	98
Comments and Comment Boxes	99
HTML Editing	100
HTML Language Support	100
HTML Popup Menu	100
WYSIWYG Editor/Viewer	100
HTML Viewing - Web Browser Interface	100
Viewing and Editing Internet Files: Installation Instructions	101
Using HTML WYSIWYG	101
XML Split Window Viewer	102
COBOL Editing	103
COBOL Lexer and DLL	103
COBOL Extensions	104
Resequence Line Numbers	104
Line Number Handling when Lines are Copied or Moved	105
Toggle COBOL Comment	105
Automatic Time/Date Stamp on Modified Lines	105
String Literals Automatically Continue on New Lines	105
Validate Line Number Sequence	106
Patch File Shows Changes in a File	106

Hex Editing	107
Insert vs. Overtyping Mode	107
Handy Hex-Editing Tips and Features	108
Clipboard and Scrap Buffers	108
Scrap Buffers	108
Multiple Clipboard/Scrap Buffers	109
Clipboard/Scrap Viewer	109
API Assistant	110
Using the API Assistant	110
Using the Checkboxes	111
API Assistant Example	111
API Assistant Databases	112
Modifying the Database	113
Automation Tools	113
Using Help in CodeWright	113
Indexing and Accessing Help Files	113
Indexing .HLP Help Files	114
Accessing MSVC 6.0 Help Files	114
Accessing Compiled Microsoft HTML (.CHM) Help Files	115
Accessing MSVC 5.0 (.IVT) Help Files	115
Accessing MSVC 4.0 (.MVB) Help Files	116
Printing	117
Print Configurations	117
Paper Selection Override	117
Color Printing	118
Print Preview	118
Multi-Copy Printing	118
Printing Line Numbers	118
Wrapping Long Lines in Printed Documents	118
Print Headers and Footers	118
 7- PROJECTS, PROJECT SPACES, AND WORKSPACES	 121
Definitions	121
What is a Project?	121
What is a Project Space?	122
What is a Workspace?	122
Creating a Project Space	122
The Project Properties List and Project Settings	124
Project Properties List	124
Default Settings	124
Working Directory	125
Creating a Project and the Members Tab of Project Properties	125
Creating a New Project	126
Adding Files to a Project	127

Adding Existing Projects to a Space	127
Auto Detect File Type to Load or Create Projects and Project Spaces	129
Reading External Makefiles and Visual Studio Workspaces	129
Steps for Setting up New Makefile/Workspace Parsers	131
Synchronize Makefile/Workspace with Project/Project Space	131
Characteristics of the File View Tab	132
Directories Tab of Project Properties	133
Storing Configuration Options with a Project	135
Reading Configuration Settings from Other Files	136
Tools Tab of Project Properties	137
Tool Categories	137
Build Tools	138
Compile Tools	139
Custom Tools	140
VCS Tools	141
Setting up Project Tools	142
Command Options on the Tools Tab	143
Filename Component Macros	144
Response File	146
Symbolic Macros	146
Errors tab of Project Properties	147
Custom Error Parsers	149
Navigating Build and Rebuild Command Output	151
Traversing the Output	152
Filters tab of Project Properties	153
Project Setup Checklist	154
Using Project Spaces	155
Selecting or Changing Projects	155
Selecting or Changing Project Spaces	155
Using Projects	156
Loading Files for Editing	156
Creating, Selecting and Saving Workspaces	156
Creating a New Workspace	157
Automatic Saving	157
Loading an Existing Workspace	158
Searching Project Space and Project Files	158
Selecting files for Check-in or Check-out	158
Project Files	159
Configuration and State Hierarchy	159
8- SET UP A COMPILER	161
Categories of Command Line Tools	161
Compiler Definition	162

Response File Contents	163
Compiler Command Line	163
Build Command Line	163
Other Tool Categories	164
Special Considerations	164
Modifying the Command Line Environment	164
Displaying Output in CodeWright (FTEE and VDOS)	165
FTEE	166
Use VDOS	166
Version Control Commands	167
9- VERSION CONTROL	169
Using Version Control in CodeWright	169
Version Control Menu	169
Source Code Revision Control - Maintenance	170
Version Control and CodeWright Projects	171
Associate Version Control Projects with CodeWright Projects	171
Add Version Control Project Files to a CodeWright Project	172
Add CodeWright Project Files to an SCC Provider Project	173
Current Project Tree List	174
VCS and the User-Defined Popup Menu	175
Modifying the Standard Popup: A Simple Example	175
Making Your Own Version Control Popup Menu	176
Using Multiple Configuration/Project Files (DOS VCS Utilities Only)	176
Version Control Integration Configuration	178
Using a Command Line Version Control Provider	178
Adding a New Command Line Provider to Version Control Setup Di-	
alog	179
Customizing Version Control Commands	179
Default Command-Line Version Control Commands Described	180
Additional Tips	182
CodeWright SCC Integration with Version Control Systems	183
Version Control for Use with a Source Code Provider DLL	183
Location of the SCC Provider DLLs	184
10- SYNCHRONIZATION	185
Initial CodeWright Setup	185
CodeWright's Synchronization Wizard, and Loading CWSync.DLL	186
Synchronization Setup From Within the Development Environ-	
ments	186
MSVC++ File Synchronization	187
VCSync Setup	187
A First View	189
Delphi File Synchronization	189

DPRSync Setup	190
A First View	191
Known Problems	191
Borland C++ File Synchronization	191
BCWSync Setup	192
A First View	193
Borland C++ Builder File Synchronization	193
BCBSync Setup	193
A First View	194
Visual Basic File Synchronization	195
VBSync Setup	196
A First View	197
TI Code Composer Studio File Synchronization	197
TICCSync Setup	198
A First View	199
Bi-directional Synchronization	199
Sync Configuration Options	199
Accessing Menu Items via Synchronization	201
11- SEARCH AND REPLACE AND NAVIGATIONAL TOOLS	203
Search and Replace and Regular Expressions	203
Search and Replace Dialog	204
Search and Replacement Edit Boxes	204
Save Settings	204
Multiple Sources Search Dialog	205
Search and Replacement Edit Boxes	205
Current Directory	206
Multiple Source Options	206
File Pattern	206
Search List	207
Search Subdirectories	207
Edit Modified Files	207
Threaded	208
Send Listing to Output Window	208
Edit Search List	208
Search Pattern	208
Patterns List	209
Drive and Directory Lists	209
Include Directory	209
List Editing Buttons	209
Default Button	209
Default Options	209
Search Direction	210
Replacement	210

Prompt	211
Search Options	212
Matches	212
Start	213
Example: Multi-Source Search	213
Fast Find on Standard Right-Click Popup	214
Incremental Searching	215
Quick Search	215
Toolbar Search	216
Regular Expressions	216
Special Characters	217
Escape Sequences	218
Matching a Character	219
Character Classes	219
Escaping Characters in a Class	219
Iteration Qualifiers	220
Examples	220
Regular Expressions: Positioning at Beginning/End of Line	221
Alternation and Grouping	221
Reference Groups and Replacement Strings	222
Placing the Cursor	223
Examples	223
Searching for Spaces, Tabs and other Blank Characters	224
Searching for Spaces or Tabs	224
Searching for New lines	224
Searching for control characters (binary/hex data)	225
Searching for New lines: Issues	225
Selective Display	226
Selective Display Options:	226
Pre-processed View	227
Viewing/Hiding Lines	227
Browse, Tags, Symbols and Objects	228
Which Navigational Tool Should I Use?	228
Browser	229
Tags	229
Outline Symbols	229
Objects Window	229
Browser Support	229
Selecting a Database	230
Traversing the Tree	230
Label Bitmaps	231
Browser Toolbar	231
Tags Support	235

Tags Setup	235
Using the Tags Database	236
Outline Symbols: Overview	236
What are Symbols?	236
Symbol Scanning	238
Outline Window	238
Outline Scanning	240
Symbol Parsers	240
Symbols Database	241
Popup Symbols Menu	242
Objects Window	242
Display an Object Hierarchy	242
Using the Objects Window	244
Objects Window and CodeSense	244
Objects Window Popup Menu	244
Symbols vs. Objects vs. Tags	246
Pros and Cons of Tags, Browsers, and Symbols	247
Bookmarks	247
Global and Local Bookmarks	247
Graphical Bookmark Images	248
Setting and Removing Bookmarks	248
Bookmarks Dialog	248
Bookmarks Window	249
Bookmarks Window: Global Bookmarks	249
Bookmarks Window: Local Bookmarks	250
Bookmarks Window: "Other Documents" Node	250
Auto-Expand/Collapse	250
Button Links	250
How it Works	250
What you See	251
Defining Buttons	251
12- CHECKING AND REFORMATTING FILES	253
Differencing	253
Interleaved Differencing	254
Difference Analysis Dialog for Interleaved Differencing	254
Interleaved Document	255
Side-by-Side Differencing	256
Difference Analysis Dialog for Side-by-Side Differencing	256
Side-By-Side Difference Window	257
Using Difference Utilities	261
Merging	262
Using the Merge Files Dialog	262
Merge Output	263

Removing Changes with Merge	264
Format Source	265
Setting up Your Formatting Criteria	265
Using the Format Feature	267
Spell Check	268
Check Spelling	268
General Tab of Check Spelling	268
Word Format Tab of Check Spelling	269
Advanced Tab of Check Spelling	271
Documents Tab of Check Spelling	272
Dictionaries Dialog	273
Dictionaries	273
13- CUSTOM INTERFACE	275
Dockable Toolbars and Windows	275
Toolbars	275
Auto-hide Toolbars	277
What Does Dockable Mean?	277
Toolbar Docking Precedence	277
Enabling and Disabling Toolbars	278
Docking and Moving Toolbars and Windows	278
Docking a Toolbar or Window Manually	278
Undocking a Toolbar or Window	279
Customizing Toolbars and Buttons	279
Adding New Toolbars	279
Adding and Changing Toolbar Buttons	280
Binding a Function to a Button	281
Combo Box History Lists	282
Editing Combo box History Lists	282
Customizing Menus	283
Menu Editor	283
Menus	283
Menu Items and Submenus	284
Changing the Functionality of a Menu Item	285
Adding a Menu Item	286
Customizing External Operations within CodeWright	287
User-Definable Popup Menus	288
Editing or Creating a Popup Menu	288
Popup Menu Semantics	291
Dynamic Menu Generation	293
Supporting Popup Menu Functions	294
Using Keymaps	294
CUA Key Commands	295
BRIEF Key Commands	295

Mouse Commands	296
Mouse Scrolling Speed	296
Inclusive or Exclusive Selection	296
Closed Selections	297
Column Marking	297
Line Selections	297
Word Selections	298
Status Line Actions	298
Text Drag and Drop	299
Mouse Copy and Move	299
Creating Windows with a Mouse	300
Drag-and-Drop File Loading	300
Expand / Collapse Selective Display	301
Reassigning Keys and Mouse Actions	301
Keymap-Specific Assignments	301
Customizing with Keybindings	301
Keystroke Recording/Playback	302
Recording a Keystroke Macro	302
Saving a Macro	302
Binding Keystrokes to Functions or Macros	303
14- FILE LOADING, BACKUP AND FTP	305
File Loading, Reloading and Validation	305
File Backups and Auto-save	307
Global Backup and Auto-Save Settings	308
Backup Settings for Specific File Types	310
Backup Settings for Individual Files	311
Formatting the Backup Specification	312
Format Controls	313
Transformation Patterns	314
Making Files Read-only	315
File Types	316
Individual Files	316
Individual File upon Opening	317
FTP: File Transfer in CodeWright	318
15- LARGE FILES	319
Swap Blocks	319
How to Change the Number of Swap Blocks	319
Block Size	320
Consider Your Resources	321
Backup Files	321
Scroll Bars	321
Pre-loading Files	322

Turn off ChromaCoding	323
Saving Large Files	324
Default File Saving Method	324
File Rewrite Save Method (for Files over 1MB)	325
16- EXTEND CODEWRIGHT	327
CodeWright API	328
Using the API from the Command Key	328
Displaying Return Values With the Command Key	328
Run Multiple Commands	329
Examples of Command Key Usage	329
Command Key Expression Evaluation	330
Macros, Macro Languages and DLL's	331
Macros and Macro Languages	331
DLL Extensions	333
Where is it Defined?	333
Perl	334
Getting Started with Perl	334
Creating and Editing Perl Scripts	335
Supplied Perl Macros	335
Perl Window	335
Popup Menu and Options	336
Online Help	336
Loading and Running Scripts	336
Running a Script Directly	337
Loading a Perl macro	337
Accessing CodeWright Functions from Perl Scripts	339
Importing Names into Perl's Namespace	339
Unloading a Perl macro	340
Using Perl's Debug Mode	340
Accessing Perl functions	340
Avoiding Ambiguity	341
Special API Functions for Perl	341
Files used by Perl for CodeWright	343
Other Perl Resources	344
AppBasic	344
Getting Started	345
Two Editors	345
Special Keybindings	346
Two Toolbars	347
Popup Menu	347
Online Help	347
UserDialog Editor	347
Object Browser	348

Creating a Macro	348
Creating a Handler	349
Object and Proc Drop-down Lists	350
Private Sub Main	351
Tips on Creating Macros	351
Creating a Modal User Dialog	352
Running the Macro	355
Creating an EventHandler in AppBasic	355
Sample EHTEST.CWB	356
Debugging Your AppBasic Macro	356
Break Points	357
Evaluate Expression and Add Watch	357
Object Browser	357
Load Macros Dialog	358
AppBasic Sample Macros	359
AppBasic-related API Commands	359
AppBasic Window Configuration	360
Example Configuration File Settings	361
Exported Functions in CWBASIC.DLL	361
API (C-like) Macros	363
API Macros Defined	363
Getting Started with API Macros	364
Creating a Macro	364
Editing a Macro	364
Special Editing Keystrokes	364
Editing API Macros in a CodeWright Window	365
Running a Macro	365
Language Definition	365
Comments	366
Identifier Naming Rules	366
Data Types for Variables	366
Declaring Variables and Arrays	366
Literal Values	368
Array Initializers	369
Automatic Type Conversion	370
Expressions	371
Statements and Statement Blocks	379
Program Flow of Control Structures	379
Run-time Error Handling	382
CodeWright Event Handling	382
Differences between API Macros and C	383
String functions	384
Making and Modifying CodeWright DLLs	385

Core Services	387
CWSTART DLL	387
CWDIALOG DLL	390
CWHELP DLL	390
Keyboard Command Sets	390
Supplemental Language Support	391
Auxiliary Services	392
Sample DLL	392
Dissecting a CodeWright DLL	393
The _init Function	393
Exporting Functions	393
Making Changes and Additions	394
Changing Existing Functions	394
Adding Your Own Functions	394
Creating New Keymap Command Sets	395
Keymap _init Function	395
Keymap Function	395
Flag Initialization	395
Basic Assignments	396
Keymap-Specific Assignments	396
Menu Accelerators	396
Recompiling a DLL	397
Using and Modifying the Makefiles	397
Adding Files	398
Compile and Link Options	398
Link Libraries	398
Microsoft Link	398
Borland Link	399
Using Your Own DLL	400
Installing Your DLL	400
Load Functions at Startup	401
Load DLL as Needed	401
17- UNIX	403
End-of-Line (EOL) Characters	403
Make UNIX EOL Characters the Default	403
Enable Auto-sense File Type Option	404
Change EOL Characters in the Source File	404
Macro for Automating UNIX EOL Conversion	405
Compiling UNIX Programs from CodeWright	406
Preserving Filename Case and File Securities between UNIX and Windows Environments	406
18- CONFIGURATION FILES & COMMAND LINE PARAMETERS	407

Configuration and State	407
Location of the Configuration File	407
Introduction to Configuration and State	408
Configuration File	408
State File	409
Other Files Containing Configuration Data	409
Example File	410
Example Interpretation	411
Processing At Startup	412
Order of Processing	412
Descriptions of Sections	413
User-Defined Sections	414
Relating Checkboxes to Functions	416
State File	417
Location of the State File	417
Contents of the State File	418
Command Line Parameters	419
Filenames	420
Parameters	420
-C<configLoc>	421
-C-	421
-G <lineNumber>	422
CW32 -g215	422
CW32 -heapalloc	422
CW32 -k mycua	422
-L <library>	423
CW32 -N	423
-P <section>	424
CW32 -s h:\home\ericj	425
Command Files	426
CW32 -s-	426
A- TAGSWNN UTILITY	427
TagsWnn Command Line Options	427
INDEX	435

Chapter Outline

The **Chapter Outline** provides an overview of the chapters in this manual.

Chapter 1: Introduction

The first chapter of this manual is a brief introduction to CodeWright. It includes a list of some of the latest and greatest features and information on where to get additional product support.

Chapter 2: Run CodeWright for the First Time

Chapter 2 describes how the initial CodeWright screen looks and briefly goes over the items on the main menu and standard toolbar. Most of the items will be given a preliminary description only, and will be described in greater detail in other chapters.

Chapter 3: Build Your First Project

This chapter quickly describes the process for building a CodeWright project within a project space. The chapter is short, intending to be a *preliminary* guide only, to the steps for building a CodeWright project. CodeWright projects and project spaces will be described in more detail in the chapter on *Projects, Project Spaces, and Workspaces*, later in the manual.

Chapter 4: Command Key, Libraries, & Environment

The **API Command**, **Libraries**, and **Environment** options on the **Tools** and **Customize** menus are used and referred to extensively throughout this manual. For this reason, they have been described as close to the beginning of the manual as possible. They are necessary and handy tools that are used for many CodeWright operations.

Chapter 5: View Setups and Language Support

Chapter 5 provides information for setting up CodeWright's visual environment, such as colors, fonts, scroll bars, and line numbers. It also goes through the steps necessary for creating and/or loading language support for languages that were previously unsupported in CodeWright.

Most of CodeWright's available language support modules are loaded out-of-the-box, but there are a few that need to be loaded manually. For those languages for which no language support modules are provided (whether loaded or not), CodeWright has two methods for creating language support. Chapter 5 describes those methods. Chapter 5 also talks about how to configure CodeWright to ChromaCode languages that are embedded in other languages (e.g. JavaScript in an HTML file).

Chapter 6: Editing & Printing

Chapter 6 provides information about the various editing features available in CodeWright, such as template expansion, file and function headers, CodeFolio Snippets, and automatic indenting. CodeWright's CodeSense and API Assistant features, utilities that insert functions with their necessary parameters and flags, are also described, along with the Clipboard/Scrap, COBOL, HTML, XML, and print features. The chapter also explains the process for setting CodeWright up to access help from different environments.

Chapter 7: Projects, Project Spaces, and Workspaces

Chapter 7 provides detailed instructions for building and using CodeWright projects, project spaces, and workspaces. The chapter describes the **Project|Properties** dialog and how external tools can be set up in the dialog.

Chapter 8: Set up a Compiler

Chapter 8 talks about setting CodeWright up to use DOS compilers. CodeWright is not a compiler, nor does it provide one. CodeWright can, however, be configured to use third-party DOS compilers. Once the compiler's command-line command is properly set, CodeWright will use the command in a DOS shell to compile the current edit buffer. When properly set up, compiles can be done from within CodeWright with the push of a button.

Chapter 9: Version Control

Chapter 9 discusses the process for using existing version control systems from within the CodeWright environment. There are 3 sections in this chapter. The first section describes CodeWright's version control interface, i.e. the menus, dialogs, and buttons that are used for performing version control operations. The second and third sections describe the two alternative methods for integrating CodeWright with version control systems: Command Line integration, or integration via the SCC Application Programming Interface (API). Which of the two methods to use depends on the type of version control system being used.

Chapter 10: Synchronization

This chapter talks about the various Synchronization programs that come with CodeWright, and how they are used. With Synchronization, CodeWright can be used in conjunction with certain GUI-based development environments, such as Microsoft Visual Studio. Starbase's Sync Technology coordinates tasks performed in CodeWright with tasks performed in the synchronized environment. It is designed to make the many features in CodeWright available to the synchronized environment. Synchronization programs are available for Microsoft Visual Studio, Microsoft Visual Basic, Borland Delphi, Borland C++ , Borland C++ Builder, and Texas Instruments Code Composer Studio.

Chapter 11: Search and Replace and Navigational Tools

This chapter has detailed information about CodeWright's Search and Replace feature including information on using regular expressions. It also describes other code-navigation tools that come with CodeWright, specifically: Tags, Symbols, the Objects Window, Selective Display, Bookmarks, and Button Links.

Chapter 12: Checking and Reformatting Files

When the editing is finished, it may be necessary to check, reformat, and/or compare/merge the finished file with previous versions. This chapter covers the tools that CodeWright supplies for this purpose. The reformat and file-checking tools include Differencing, Merging, Format Source, and Spell Check.

Chapter 13: Custom Interface

Chapter 13 describes CodeWright's toolbars and talks about modifying existing keystrokes; modifying, adding, and removing menu items and toolbar buttons; and making keystroke macros. It briefly describes the keymap-choices that are available in CodeWright, the main ones being CUA, BRIEF, Epsilon, and vi. It also describes some of the mouse actions CodeWright has that are not available in most other editors, such as column selection and text dragging and dropping.

Chapter 14: File Loading, Backup and FTP

Chapter 14 talks about CodeWright's Autosave, File Backup, and FTP features. It also discusses the file-validation and file-preloading features, which are designed to prevent file-loss and/or file-corruption.

Chapter 15: Large Files

The steps for preparing CodeWright to handle files as large as 2 Gigabytes are covered in this chapter.

Chapter 16:Extend CodeWright

CodeWright is a fairly complete product. There is not much that it lacks. Nevertheless, there always seems to be one job for which necessary tools just can't be found. If this is the case, CodeWright has some tools for extending the editor. These tools consist of three macro languages (Perl, AppBasic, and API Macros), and DLL-source code. They are provided in the event that CodeWright doesn't already have the necessary functions for handling the task at hand. These tools can be used together (i.e. a macro that uses a function from a DLL or another macro, or vice versa), or on their own. DLL Add-Ons can be written in any programming language, and they can be loaded interactively from within the editor. Chapter 16 describes these tools in detail.

Chapter 17: UNIX

CodeWright is a Windows-only product. That is to say that it can't be installed on any operating system other than Windows (NT 4.0, 95, 98, 2000, ME). Even so, it is possible to edit UNIX files in CodeWright. The main difference between files edited on UNIX systems and files edited on Windows systems are the characters that make up the ends of lines. Chapter 17 describes CodeWright features for automatically detecting and inserting the proper End of Line characters. It also provides other information about editing UNIX files in CodeWright.

Chapter 18:Configuration Files & Command Line Parameters

There are a number of files CodeWright uses for maintaining configuration information. These files are stored by default in CodeWright's installation directory (project files being the exception), but they can be stored in any directory desired. The files are listed and manipulated in CodeWright's **Project|Properties|Directories** dialog, which is described in the chapter on *Projects, Project Spaces, and Workspaces*. Chapter 18 briefly describes the two main configuration files used by CodeWright, CWRIGHT.INI, and CWRIGHT.PST. It also describes command line parameters that can be used for specifying different configuration files, different files, and different CodeWright states (among other things), to load when the CodeWright executable loads.

Appendix A: TagWnn Utility

TAGSWnn (TAGSW16 or TAGSW32, depending on your platform) is the Tags program supplied with CodeWright. It automatically generates a tags database and compiles it into a format that can be used by CodeWright's built-in browser. The program is run when **Build Tags** is selected 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 wish to know more about the program, its options are briefly described in this appendix.

Chapter 1

1- Introduction

CodeWright is an editor with features geared towards software programmers. It is designed to support multiple programming languages in that (at a minimum) it ChromaCodes™ the syntax elements of the language. It is the first professional-quality, extensible programmer's editor written for Windows from the ground up.

What's In This Chapter

This chapter introduces CodeWright and lists some of the features that make it a great tool for editing needs. It specifically covers the following items:

- What makes CodeWright different.
- Additional product support.
- A list describing some of CodeWright's key features.

What Makes CodeWright Different

Listed below are some of the things that make CodeWright different from other editors:

- CodeWright is from Starbase Corporation, which ensures guaranteed satisfaction and the best support available.
- CodeWright is not a port of a program from DOS, UNIX, or any other operating system.
- CodeWright is the innovator. Here are just a few of the innovations CodeWright brought to programmer's editors under Windows:
 - ✓ Syntax Highlighting (ChromaCode)
 - ✓ DLL extensibility
 - ✓ Merge and Difference
 - ✓ Elided text (Selective Display)
 - ✓ Help Manager

- ✓ IDE integration
- ✓ API Assistance
- ✓ Button Links
- ✓ HTML viewer
- ✓ MVB, IVT, and HTML help file support along with traditional HLP help file support.

Additional Product Support

If a problem arises in using or programming CodeWright, there are a number of resources you can use. First, check the README.TXT file that was shipped along with CodeWright. It is placed in CodeWright's home directory during installation. It will warn you about any known "gotchas" that are not covered by the manuals. The next places to look for help are in the *Getting Started* manual, this *User's Guide*, and the online help files.

Web Page

You can access our home page on the Internet's World Wide Web using the following URL: <http://www.starbase.com>. Information and services available there include:

- messaging to sales and product support
- pricing and product descriptions
- problem and enhancement report forms
- downloading of Add-Ons and patches

Internet Mail

You can send email to codesupport@starbase.com, and we will assist you as soon as practical. For sales matters, use the address codesales@starbase.com.

Fax

A fax will normally get a quick response. Our fax number is (503) 641-6001.

Phone Support

If you urgently need some information in order to continue using CodeWright, or if you have an urgent bug report, give us a call. The phone number is (503) 207-1190.

Key Editor Features

The following list describes some of the items CodeWright provides for making the task of editing easier. Combined, they make CodeWright the most powerful Windows editing tool on any platform.

Key Editing Features	
Category/ Chapter	Feature
Introduction	Flexible multiple-window, multiple-file interface. Edit as many files in as many windows as your resources allow.
Run CodeWright for the First Time	Configuration Wizards: to assist you in setting up the Help Index file, setting up synchronization, etc. Includes an Answer Wizard.
	Support available for Microsoft's HTML Help Viewer .
	Customization Shortcut on the Status bar allows you to quickly access many of the items you'll want to configure.
	Document/Window Manager has a Window tab that consolidates settings that affect individual documents and windows.
	Wrap option on the Display tab of the Document/Window Manager allows you to wrap text on the <i>display only</i> , 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.
	New Forward and back buttons on the standard toolbar allow documents and document positions to be navigated based on a history of the documents and positions that have been accessed during the current CodeWright session.

Key Editing Features	
Category/ Chapter	Feature
View Setups and Language Support	Tile specified windows horizontally or vertically, all in one row, from the Document/Window Manager or globally from Customize Environment .
	ChromaCoding Lexers: Interactively create ChromaCoding lexers to color the syntax for your programming language.
	New Embedded language support. Configure CodeWright to ChromaCode languages that are embedded in other languages. Set a different background color for embedded language blocks.
	New Split Windows. Windows can be split up to 4 ways using the appropriate CUA key sequence, or using splitter 'notches' that are available on vertical and horizontal scrollbars.
	New Support for keywords that can be defined by the user. For example, XML allows users to define unique keywords for their projects. CodeWright has an option and controls for parsing and coloring those keywords.
	New New ChromaCoding lexers for XML, ASP, PHP, Cold Fusion Mark up Language, Cold Fusion script, and others.
	View Setups for storing sets of colors, fonts and window attributes under individual names.
	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.
	ChromaCode™ use of color to signal changed lines or highlight language syntax.

Key Editing Features	
Category/ Chapter	Feature
Editing & Printing	Multiple Clipboards/Scrap Buffers allow copying more than one item for later use without overwriting previous items. An auto-increment clipboard option increments the clipboard/scrap buffer to be used, just prior to copying or cutting the new content.
	New CodeSense for Java.
	Clipboard and Scrapboard viewer is 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 C/C++ and Java source files, to provide word completion for symbols; function parameter help; and symbol definition help.
	New COBOL Extensions: Automatically insert COBOL line numbers, toggle comment delimiters, insert time/date stamps to modified lines, more.
	CodeFolio Snippets: New CodeFolio tab on the Project Window. Provides a directory of code samples, specific to programming languages, which may be inserted into the current document. User may add Snippets to the directory.
	Zoom functionality is now available on the Print Preview dialog.
	Hex mode for editing binary files, including inserting and deleting bytes.
	Unlimited Undo and Redo of commands (from the Edit menu).

Key Editing Features		
Category/ Chapter	Feature	
		Powerful Templates for language constructs, personalized function headers and more. Any CodeWright API function that can be executed interactively can be executed within a template.
	New	WYSIWYG HTML editing with a toolbar for quickly inserting graphics, tables, and commonly used tags and attributes.
		API Assistant helps you accurately complete function calls by showing you the types and options for each parameter defined for the function. Check the desired boxes; it then constructs the proper call for insertion into your text.
Projects, Project Spaces and Workspaces		Project spaces: Multiple projects may be displayed at a time within a project space.
Synchronization		Bi-directional Synchronization with Microsoft Visual Studio and Texas Instruments Code Composer Studio (files in CodeWright will automatically open in the other environment using the bi-directional sync buttons on the MSDevSync and TICCSync toolbars).
	New	Synchronization with Texas Instruments Code Composer Studio.
		Special synchronization with Compiler Environments including Borland's C++ IDE, Microsoft's VC++ Workbench, Borland's Delphi, Borland's C++ Builder, Visual Basic 6.0 and Texas Instruments Code Composer Studio.
Search and Replace and Navigational Tools		Save and restore cursor position when closing and re-opening CodeWright.

Key Editing Features	
Category/ Chapter	Feature
	Selective Display mode for selecting which lines to make visible or invisible. Allows selecting lines to be viewed with grep-like commands, or by preprocessing #ifdefs – you name it.
	New Objects tab on the Project Window displays a hierarchical view of 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. It can be used to view and browse code.
	New Preserve and restore selective display in all open documents when closing and re-opening CodeWright.
	Complete support of regular expressions in searching and replacement.
	Multi-document or file-based search and replace, 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 To Do list. These Links appear as graphical buttons within the text file.
	Several types of browsing, including Tags, Microsoft .BSC files, and Outline Symbols.
	Unlimited local and global bookmarks.
Checking and Reformatting Files	Expanded spell check dialog and additional dictionaries for languages other than English.
	The spell checker will check the spelling of source code comments and strings only, if desired. Spell check the whole buffer, or just the selection.
	Side-by-side Differencing, Difference Editing, and Merging of files.

Key Editing Features	
Category/ Chapter	Feature
Custom Interface	Auto-hide toolbars, and auto-hide menus in Full Screen mode.
	New Toolbar docking precedence. Allow toolbars to take advantage of up to two full horizontal or vertical edges of the CodeWright window.
	Interactively definable Toolbars for quick mouse access to common commands.
	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 , Backup and FTP	Auto-save limit: Allows you to limit the size of files that are auto-saved.
	Additional auto-save features on the Environment Backup dialog for controlling the directory and file extension to be used for auto-saved files.
	FTP: Transfer files from within CodeWright. The FTP feature supports file structures for UNIX, Tandem Guardian, and VMS hosts.
	Optional auto-save at scheduled intervals or during idle periods.
Large Files	Line length and file size virtually unlimited.
Extend CodeWright	Enhanced API Macro Language.
	CUA, BRIEF, Epsilon, or vi style key commands.
	Completely remappable keyboard – Create your own command set.

Key Editing Features	
Category/ Chapter	Feature
	New API macros support arrays.
	Configurable and <i>Extensible</i> through Macros and DLLs.

2- Run CodeWright for the First Time

This section describes the way CodeWright looks when it is first loaded, giving some tips about things to know before getting started. Many of the items described will be deferred to other chapters in this manual for more information. The descriptions provided are brief, intended only to familiarize the user with the initial CodeWright environment.

Configuration Wizards

If the option to load CodeWright after installation is chosen, the CodeWright screen will display with the Configuration **Wizard Choices** on top. Configuration Wizards assist with various jobs in CodeWright. They walk through the steps necessary for some of the more complicated configuration tasks. The **Wizard Choices** include:

- The **Answer Wizard**: Displays lists of help topics in response to queries entered in the form of questions.
- The **Help Index File Wizard**: Goes through steps for configuring the help index file. For more information about configuring help, see the topic *Using Help in CodeWright* in the chapter on *Editing & Printing*.
- The **Sync Technology Wizard**: Goes through the steps needed to configure CodeWright to synchronize with selected development environments. Supported environments are:
 - ✓ Microsoft Visual Basic (V6.0)
 - ✓ Microsoft Visual Studio (V5.x or V6.x).
 - ✓ Borland C++ (V5.0)
 - ✓ Borland C++ Builder (V4.0 and V5.0)
 - ✓ Borland Delphi 32 (V4.0 and V5.0)
 - ✓ Texas Instruments Code Composer Studio (V1.0 and V1.1)

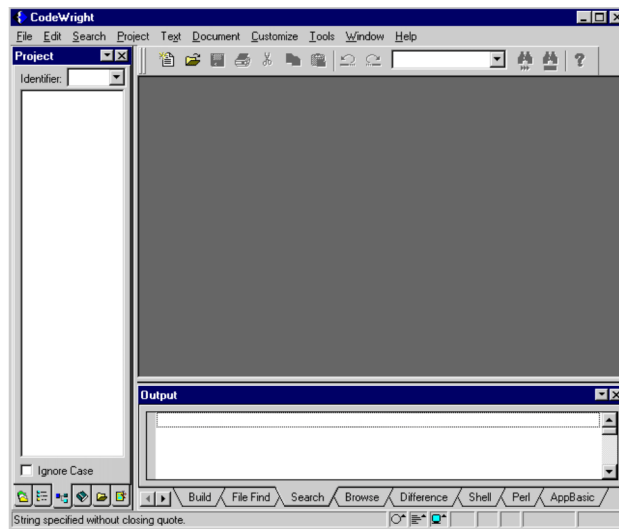
For more information about CodeWright's Sync technology, see the chapter on *Synchronization*.

- The **CodeSense Libraries Wizard**: Brings up the **CodeSense Global Configuration** dialog where libraries that point to designated C/C++ and Java files are created. The libraries are used by CodeWright's CodeSense feature. When properly configured, the CodeSense feature provides advanced syntax help and word completion for code that is being typed. See the chapter on *Editing & Printing* for more information.

A First Look

After completing any necessary configuration tasks, and accessing needed help tips, the **Configuration Wizard** and **Tip of the Day** dialogs can be closed. The resulting CodeWright screen displays a menu bar at the top, with the **Standard** toolbar docked directly beneath. The window docked on the left is the Project Window, and the window docked on the bottom is the Output Window. All of these items will be described briefly in the remainder of this chapter.

Initial CodeWright Screen



Two files open by default with the initial CodeWright session: README.TXT and UPDATE.TXT. README.TXT contains important information about the latest release, and UPDATE.TXT lists the latest features. It's a good idea to read through these documents before closing them.

The Menu Bar

The menu bar consists of ten items, described in the following topics.

File Menu

The first menu is the **File** menu. The **File** menu is intended for standard file operations, i.e. **New**, **Open**, **Save**, **Print** etc. Additional items are described (or deferred to other chapters for more information) below:

- **Difference and Merge.** The **Difference** and **Merge** utilities are described in detail in the chapter on *Checking and Reformatting Files*.
- The **Find** menu-item is used for finding files on any storage medium. Standard DOS wildcard characters can be used when searching for files with this option.
- **Send Mail and Reload.** **Send Mail** sends files via MS mail or cc:Mail, and **Reload** reloads the current file from disk.
- **Change Directory** accesses a dialog that allows the current directory to be changed.
- **Filters.** The **Filters** dialog defines file specifications or wildcard patterns, such as *.c, to filter files of interest from otherwise long lists of files. It's a good idea to set the filters during the initial CodeWright session so that they'll be ready to use in the future. Here are a few useful filters:

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

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

- **Print and Print Setup.** The last two items on the **File** menu, **Print** and **Print Setup**, are described in more detail in the chapter entitled *Editing & Printing*.

Note: Loading a library in CodeWright's **Customize|Libraries** dialog optionally turns on an additional **FTP** item on the **File** menu. The **FTP** item can be used for transferring files from CodeWright to remote hosts. For more information on CodeWright's FTP feature, refer to the chapter on *File Loading, Backup and FTP*.

Edit Menu

The first two sections of the **Edit** menu contain standard **editing** items, such as **Undo**, **Redo**, **Cut**, **Copy** and **Paste**. Three items that may not be familiar are the **Scrap Buffer**, **Append**, and **Erase** items:

- 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.

CodeWright's Scrap and Clipboard features are both described in more detail in the chapter on *Editing & Printing*.

The third section of the **Edit** menu has options for inserting items into the current buffer:

- **Insert File** brings up a dialog for selecting a file to be inserted into the current document.
- **Insert Literal** 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.
- **Insert Link** inserts a button link at the cursor position in the current document. **Button links** are special action buttons that CodeWright lets you embed in your text files. You may use them to view bitmapped images, bring up a related document or spreadsheet, run a macro or just to make notes. **View Links** (the last item on the **Edit** menu) is used for viewing the contents of the **Button Links** database.

Button Links are described in more detail in the chapter on *Search and Replace and Navigational Tools*.

In the Macros section of the **Edit** menu, **Record**, **Playback**, **Keystroke Macros**, and **Run Key Macro** are for making, testing, editing and using Keystroke Macros. CodeWright's Keystroke Macros are described in greater detail in the chapter on *Custom Interface*.

Search Menu

The **Search** menu contains all the items necessary for performing search and replacement operations on single or multiple files. All of CodeWright's search features are described in detail in the chapter on *Search and Replace and Navigational Tools*. The one item worth describing here is **Options**. **Search|Options** brings up the **Default Search and Replace Settings** dialog, where you set up searches to operate the way you expect.

It is prudent to set 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 it be momentary or continuous? If continuous, should **Restrict to Selection** be turned off so that **Search Again** will work as expected?
- Should the word that the cursor sits on be in the **Search** dialog by default, or should the word be typed?

Project Menu

The **Project** menu is used for creating, opening, closing and manipulating projects, project spaces, and workspaces:

- Projects are used to organize and store individual sets of files and configuration settings. A project, at a minimum, is a list of files that you find it useful to group together logically.
- Project Spaces organize sets of projects. Each project must be part of a project space. If an existing project is opened without first creating a project space, a project space will automatically be created to envelop the project.
- Workspaces preserve options and settings for the currently open windows and documents; this can include project and non-project files. Reload the workspace to pick up where you left off with those files.

Options on the **Project** menu include:

- **Project Space**, which displays a submenu containing **New**, **Open**, and **Close**, for creating, opening, and closing project spaces, respectively. The submenu also contains options to add or remove projects from the project space. **Project|Set Current** is for opening a project within a project space.
- **Compile**, **Compile(Debug)**, **Build**, **Rebuild**, **Rebuild (Debug)**, **Debug**, and **Execute** have to be configured in the **Project|Properties** dialog in order to use them. More information about CodeWright projects are provided in the chapters on *Build Your First Project* and *Projects, Project Spaces, and Workspaces*.

Text Menu

The **Text** menu has various items for formatting and editing text. Some of these items, like **Selective Display**, the **Comment** items, and **Format Source**, are discussed in-depth in various chapters. The other items are fairly self-explanatory and will be briefly described here.

- **Word Wrap** enables and disables automatic line wrapping. The lines will wrap at the right-margin mark, which is set on the **General** tab of the **Tools | Customize | View Setups** dialog. On that dialog, if **Wrap Display Mode** is selected, but **Wrap Column** is not, the right-margin will be automatically adjusted when the window is resized.
- The **Upper** and **Lower** menu items will respectively capitalize or un-capitalize selected text.
- The **Slide In** and **Slide Out** menu items will slide the current line or the current selection right (in) or left (out) the length of a tab stop. The **Prompted Slide In/Out** items prompt the user for some text that will be used to slide the text in or out.
- The **Left Justify**, **Right Justify**, and **Center** items will position text within a column or line selection as their names describe.
- The **Enumerate** menu item inserts line numbers in ascending or descending order for selected text, and **Format Columns** aligns columns within a selected block.

Document Menu

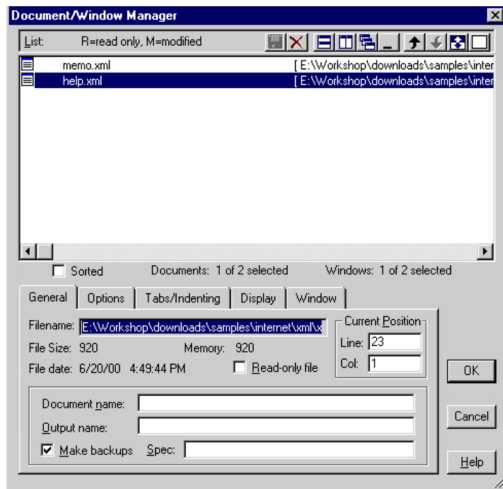
The **Document** menu has items that access dialogs and perform operations that are commonly used on documents. The first and second sections contain fairly self-explanatory items for creating, closing, clearing and maneuvering documents. The third section has items for inserting and moving between bookmarks. Bookmarks are described in detail in the chapter on *Search and Replace and Navigational Tools*.

The last item on the **Document** menu accesses the **Document/Window Manager** dialog. The **Document/Window Manager** dialog lists documents and windows that are currently open in CodeWright. It can be used to change the attributes of the open windows and documents on an individual basis.

The **Document/Window Manager** dialog contains a window listing the files and windows currently open, within which files can be selected for manipulation. The window's file-icons represent the different document/window-states that the files are in (as described in the online help topic *Document/Window Manager Dialog*), as well as their read-only or modified status.

All operations performed from **Document/Window Manager** can be performed on a per-document or per-window basis.

Document/Window Manager



Some of the things that can be done in **Document/Window Manager** include:


- Specifying indent options.
- Tiling/cascading individually selected documents or windows.
- Setting different **View Setups** (fonts, colors, etc) for individual documents/windows.
- Changing display modes (e.g. **Selective** or **Hex** displays).
- Changing window attributes (e.g. scrollbars, line numbers).
- Setting soft line **Wrap** (located on the **Display** tab) - Wraps lines without inserting a line feed, adjusting 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.

Customize Menu

Use the items on the **Customize** menu to configure various parts of CodeWright.

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

Portions of the Environment dialog are discussed in the chapter *Command Key, Libraries, & Environment*. However, since the tabs in the dialog relate to a wide variety of features in CodeWright, the items in the dialog will be covered throughout this manual.

- The **Read Configuration Data** 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 CodeWright how it should look and behave.
 - The **Libraries** option on the **Customize** menu is used for interactively loading CodeWright DLLs (or Add-Ons) to extend the functionality of the CodeWright program. A number of commonly used Add-Ons are contained in the list of CodeWright Libraries on the **Load CodeWright DLL** dialog. CodeWright Libraries can be quickly and conveniently loaded by placing checkmarks next to the ones to be loaded. More Add-Ons are available on Starbase's WebPages, at <http://www.starbase.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 the chapter on *Custom Interface*.
 - The **View Setups** option accesses a dialog used for viewing and changing view setups which store sets of document and window preferences, such as colors, fonts, line numbers, and scroll bars.
 - The **ChromaCoding Lexers** option accesses the **ChromaCoding Lexer Settings** dialog. Here you can create or modify lexers to color the various elements of your programming language "vocabulary". The vocabulary of a language includes identifiers, braces, preprocessors, keywords, operators, strings and comments.
 - The **Language** option accesses a dialog that is used for controlling language-specific features, such as ChromaCoding (syntax coloring), and template expansion.
- Note:** View Setups, ChromaCoding Lexers, and Language are described in the chapter on *View Setups and Language Support*.
- The **CodeSense Global Configuration** option accesses the **CodeSense Global Configuration** screen. CodeSense will complete function names as they are being typed and automatically suggest appropriate parameters to be inserted for the function. It is available for C/C++ and Java files. CodeSense works by parsing the programming libraries that you add on the **CodeSense Global Configuration** screen, as well as project source files and files that are open in CodeWright. CodeSense must also be configured on the **Customize|Language|CodeSense** dialog to be used. For more information on CodeSense, refer to the chapter on *Editing & Printing*.
 - An additional **CobolExt Settings** item is available on the **Customize** menu when **COBOL Extensions** is loaded in **Customize|Language**. See the topic *COBOL Extensions* in the chapter *Editing and Printing*.
 - ✓ Many of the items available on the **Customize** menu may also be accessed directly by right-clicking on the Customization shortcut icon  on the status bar.

Tools Menu

The **Tools** menu has a variety of tools for manipulating and extending CodeWright. Items on this menu are described/referenced below:

- The first five items on the **Tools** menu, (with the exception of **API Command**), are for CodeWright's macro languages:
 - ✓ The **API Macros** and **Run API Macro** menu items are used for making and running CodeWright 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 greater detail in the chapter on *Extend CodeWright*. (The **API Command** option is described in the chapter on *Command Key, Libraries and General Environment*.)

- The **Version Control** item on the **Tools** menu accesses a submenu containing items that either run version control operations or that access dialogs used for running or setting up version control in CodeWright.

When looking at this submenu, it is important to remember that CodeWright 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 CodeWright can be found in the chapter on *Version Control*.

- The **Spell Check** option on the **Tools** menu accesses CodeWright's spell checker. CodeWright's spell checker is described in more detail in the chapter on *Checking and Reformatting Files*.
- The next three items before **Customize** are handy utilities for performing sundry tasks:
 - ✓ The **Filter** option brings up a dialog that lets 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 last section of the **Tools** menu is optional. It contains tools that may vary if CodeWright has been customized. Custom tools are set up in the **Project|Properties** dialog on the **Tools** tab. These are described more comprehensively in the chapter on *Projects, Project Spaces, and Workspaces*.

Window Menu

Similar to the **Document** menu, the **Window** menu is used for manipulating and creating windows opened in CodeWright. It is important to understand how CodeWright differentiates "windows" and "documents". Depending on the configurations that have been made in CodeWright, the files that are open may not always be contained in their own windows. The difference between windows and documents is described below.

The Difference between Windows and Documents

Some confusion may arise when attempting to discern how CodeWright differentiates "windows" and "documents". Depending on the keymap being used, documents open in CodeWright may not all appear in their own window.

- Most of the keymap emulations that CodeWright provides (e.g. BRIEF, Epsilon, and vi) use one window for all documents by default. While there may be 10 documents open, only one of them will be seen at a time in the one window that is open. This characteristic defers to the DOS days of the editors from which the referenced keymap emulations were derived.
- The CUA keymap, however, uses one window for every document. Therefore, in CUA, each window will contain its own document.

The "document per window..." characteristic can be toggled for any keymap using **One Document per Window** in **Customize | Environment | General**. Keymaps are described in the chapter *Custom Interface*.

Details of the Window Menu

The items on the **Window** menu are fairly self-explanatory:

- **New Window** creates a new window. If a document is open, the new window will contain a second instance of that document.
- **Full Screen** puts documents into full-screen mode, eliminating the clutter caused by menus and toolbars, maximizing editing space.
- **Tile** and **Cascade** are alternative ways to arrange CodeWright windows.
- **Arrange Icons** arranges minimized windows at the bottom of the CodeWright 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 *Project and Output 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**, see the topic *Document Menu*, in this manual.

Help Menu

CodeWright's **Help** menu is for accessing and configuring CodeWright's online help and API Assistant. More information on Help and the API Assistant can be found in the chapter on *Editing & Printing*.

Output and Project Windows

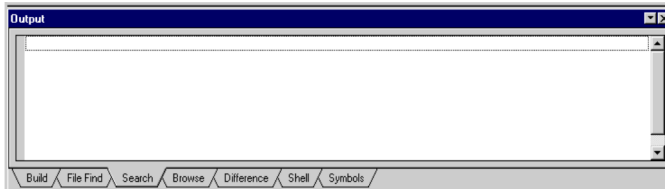
The default CodeWright screen has two system windows, one docked on the bottom, and one on the left. The window docked on the bottom is called the Output Window. The window docked on the left is called the Project Window. These two windows can be "undocked" by dragging them away from any edge of the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

- Further information on customizing the Output and Project Windows can be found in the chapter on *Custom Interface*.
- The Output Window is described to a greater extent in various chapters of this manual, as it contains a number of tabs that relate to various features in CodeWright.
- The Project Window is described in greater detail in the chapter on *Projects, Project Spaces, and Workspaces*.

Output Window

The Output Window initially has seven tabs and is docked on the bottom edge of the CodeWright screen. Three more tabs can optionally be added 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.

The Output Window



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

The default tabs of the Output Window are:

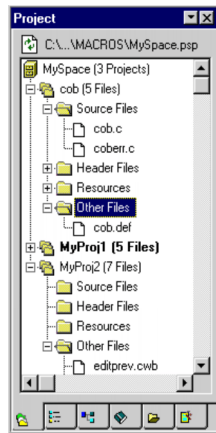
- **Build** tab: Displays the results of builds or compiles.
- **File Find** tab: Displays the results of file searches produced by **File | Find**.

- **Search** tab: Displays the results of string searches produced from search operations.
- **Browse** tab: Displays tags or browser database files.
- **Difference** tab: Displays the differences of two files, side-by-side.
- **Shell** tab: Acts as a virtual DOS shell.
- **Symbols** tab: Allows you to navigate through files, similarly to the **Browse** tab.

The Project Window

The Project Window is initially docked on the left edge of the CodeWright screen.

Project Window



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

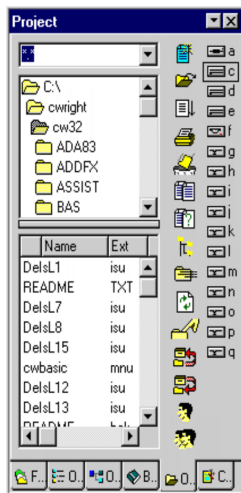
- **File View** tab: Lists the files in the current project in hierarchical form, and lets you operate on individual files or groups of files. More information about the **File View** tab can be found in the chapter on *Projects, Project Spaces, and Workspaces*.
- **Outline** tab: Presents a hierarchical 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 the chapter on *Search and Replace and Navigational Tools*.
- **Objects** tab: The Objects tab displays a hierarchical view of C/C++ and Java objects/symbols. The objects/symbols displayed are associated with the name that is typed in the Identifier box at the top of the window. The window can be used to view and browse code. More information about the Objects tab can be found in the chapter on *Search and Replace and Navigational Tools*.

- **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 the chapter on *Search and Replace and Navigational Tools*.
- **Open** tab: Acts like a persistent File Open dialog and file manager in one. It presents a list of icons representing valid drives from which to choose, a directory tree, a place to specify a file filter to limit the files displayed, and a box where 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: *More on the Open tab*.
- **CodeFolio** tab: Presents a directory tree of pre-defined chunks of text that can be specific to programming languages. The text can be inserted into your current document. Custom and user-defined Snippets can also be added to the window. Refer to the topic *CodeFolio Snippets* in the chapter on *Editing & Printing* for more information.

More on the Open Tab

The Project Window's **Open** tab is a handy tool for file-operations. Opening files is just the beginning of what it can do. There are a total of 20 other operations that can be performed, as represented by a series of icons on the tab's frame. Tool tips (small popup messages) describe each of the icons.

Project Window: Open Tab

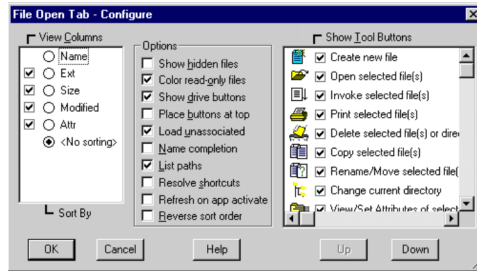


Use the **File Open Tab - Configure** dialog to remove, add, or reconfigure the icons on the **Open** tab. To access this dialog, click on the **Configuration Dialog** button




The following displays:

File Open Tab – Configure



Two of the icons you can set for the **Open** tab are for user-defined commands: 

and . User-defined commands can be set to perform any operation on files selected within the window. The user commands must be CodeWright API function calls, rather than DOS commands or the like. You may, however, execute DOS commands by specifying the API function `ExecUserCmnd` (help on `ExecUserCmnd` can be found in CodeWright's online help).

Example:

```
ExecUserCmnd "Dir"
```

The **File Open Tab - Configure** dialog is also used to set whether or not to display hidden files, 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 Standard Toolbar

CodeWright has several toolbars, but only the **Standard** toolbar is turned on by default. Since the **Standard** toolbar is part of the default CodeWright screen, it will be introduced here; all of the toolbars are described in more detail in the chapter on *Custom Interface*.

Standard Toolbar



The Standard toolbar consists of the following:

- The first two buttons on the standard toolbar are forward and back buttons. They allow documents and document positions to be navigated based on a history of documents and positions that have been accessed during the current CodeWright session. Drop-down lists are available for each button. The drop-down lists allow specific documents to be accessed without having to cycle through all the documents in the list.
- The nine buttons that follow the forward and back buttons are for editing and file operations: creating, opening and saving files; printing; cutting, copying and pasting text; and redo/undo of the last change made.
- The next item on the **Standard** toolbar is the **Toolbar Search Box**. 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, and press .
- The next two buttons on the **Standard** toolbar are also related to searching:
 - ✓ Press  to repeat the last search
 - ✓ Press  to search for the word under the cursor in the current document.
- The last button on the **Standard** toolbar (the question mark) is for accessing CodeWright's online **Help**.

3- Build Your First Project

This chapter provides a quick guide for building CodeWright projects and project spaces. Projects and project spaces are covered more fully in *Projects, Project Spaces, and Workspaces*.

CodeWright projects offer a means for storing related files as a unit. They make version control operations simpler and more convenient, and they store unique configuration settings, like compiler commands, with each project.

Project spaces are an extension of the project facility. They store sets of projects, allowing multiple projects to be displayed at a time. A project space must be open before a project can be created.

Making CodeWright Projects and Project Spaces

The appropriate order for making projects is as follows:

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, do the following:

1. Click **Project|Project Space|New**.
2. Click **Browse** to set the directory in which the project space will be stored.
3. Name the project space. Project space configuration files use .PSP extensions.

Note: The chapter *Projects, Project Spaces, and Workspaces* describes the **Look in directory for external workspace** option.

4. 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. The information about creating projects and adding files is contained in the topics *Creating a Project* and *Adding Files to a Project*. But before the project is created, it is necessary to set the project defaults, as explained in the next section.

Setting Project Defaults

Configuration settings for new projects are inherited from the **<Default Settings>** item in CodeWright's **Project Properties** dialog. Changes made to the settings while a project is selected will be stored with that project only. Closing the project will cause all settings to revert to the defaults. For this reason, it may be a good idea to set some configurations for **<Default Settings>** before the project is created, so that the settings won't change when the project is closed. For example, it may be wise to set up the compiler before creating a project so that the settings won't be lost when the project is closed. (Information for setting up a compiler can be found in the chapter *Set up a Compiler*. Information on project settings can be found in the chapter on *Projects, Project Spaces, and Workspaces*.)

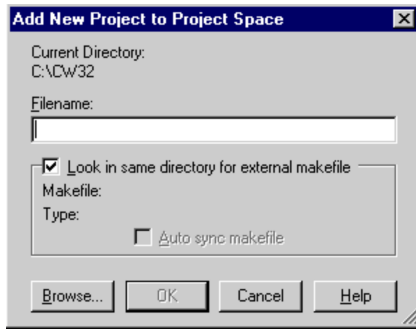
To make default settings, highlight **<Default Settings>** in the **Project| Properties** dialog, and then make various configuration settings in the different tabs of the dialog. Once desired **<Default Settings>** have been established, appropriate modifications can be made for projects on a per-project basis.

Creating a Project

Once a project space has been created, individual projects can be created for the space. Projects are created in the **Project|Properties|Members** dialog. This dialog should appear immediately after a new project space has been created. To otherwise access the dialog, select **Project|Properties**, and then click the **Members** tab. Carry out the following steps to create a CodeWright project:

1. Highlight the current project space in the listbox on the left.
2. Click  to access the **Add New Project to Project Space** dialog. (The same dialog can be accessed using the **Add New Project** item on the **Project|Project Space** menu.)
3. In the **Add New Project to Project Space** dialog, click **Browse** to make sure that the directory the project is being created in is appropriate, then name the project.





Add New Project to Project Space



4. Click **OK**. CodeWright creates a new configuration file in the chosen directory with the name that was typed and a .PJT extension.

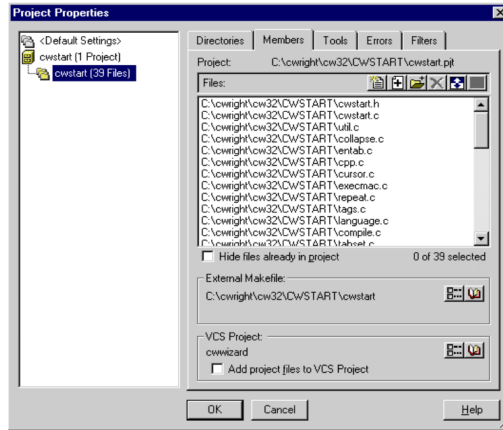
Adding Files to a Project

Files can be added once the project has been created. Complete the following:

1. Access the **Members** tab of the **Project|Properties** dialog.
2. Highlight the project in the listbox on the left.
3. Add the files using one of the following methods:
 - Click  to access the **Scan Existing Files into Project** dialog. Add individual files by typing in the name or add multiple files using file filters (e.g.*.C). Paths are optional (e.g. C:\TEST*.H). Use the [...] button to browse for different directories.
 - Multiple filters can be used by separating each filter with semi-colons (e.g. *.C;*.H;*.CPP). The **Include Subdirectories** option causes files in subdirectories of the current directory to be included with files that are added.
 - Click  to access the **Select One or More Files to Add to Project** dialog. Add individual files by double clicking or typing the name. Add multiple files by  or  clicking.
 - Use the **External Makefile** option to read files from an existing makefile. See *Reading External Makefiles* in the chapter *Projects, Project Spaces, and Workspaces*.
 - Use the **VCS Project** option to read files from a version-control-project-file (see *Using Version Control in CodeWright*, in the Chapter *Version Control*).

Files that are included in the project are listed in the **Files** listbox in the center of the dialog. Files can be deleted from the project by selecting them in the listbox and pressing the **Delete** button.

Project Properties Members



Project|Properties

The **Project|Properties** dialog is where all project configurations are set. Most of the settings made in this dialog can be used without an open project, in which case they would be global settings and would become the defaults. More information on the **Project|Properties** dialog is provided in the chapter on *Projects, Project Spaces, and Workspaces*.

Project Tools

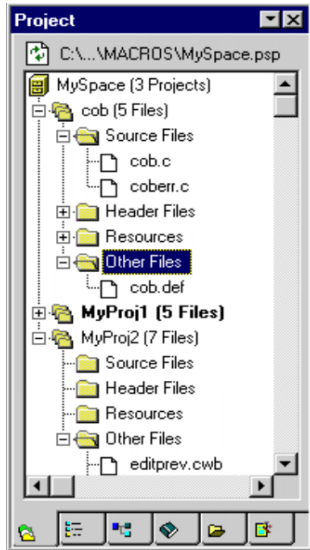
Once the necessary files have been added in the **Members** tab of the **Project|Properties** dialog, the project space and projects are basically complete. Before clicking **OK**, though, it often desirable to set up some tools in the dialog's **Tools** tab. See the topic *Tools Tab of Project Properties* in the chapter *Projects, Project Spaces and Workspaces* for more information.

Project Window File View

Once the project space has been populated with all the appropriate projects, and the projects have their necessary files and configurations, the **OK** button can be pressed for the **Project|Properties** dialog. The project space is then created.

For a graphical display of the project space, click the **File View** tab of the Project Window. The project space and its included projects will be displayed. The projects can be further expanded to display each project's member-files. An example of the **File View** tab, expanded to show project space, projects and files, follows:

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.

For more information on CodeWright projects, see the chapter on *Projects, Project Spaces, and Workspaces*.

Chapter 4

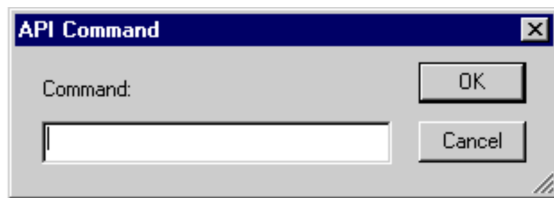
4- Command Key, Libraries, & Environment

The **API Command**, **Libraries**, and **Environment** options on the **Tools** and **Customize** menus are used and referred to extensively throughout this manual. For this reason, they are briefly described here. They are necessary tools that are used for many CodeWright operations.

API Command Dialog/Prompt


Select the **API Command** item on CodeWright's **Tools** menu to bring up a prompt that interactively runs CodeWright API functions.

API Command





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

CodeWright has numerous API functions, many of which are available for interactive use from within the CodeWright editor. CodeWright APIs can be used to change keystrokes, buttons, and menu commands; they can be used in the macros and DLLs that extend CodeWright; and they can be used interactively from the **API Command** prompt.

Do the following to use the **API Command** menu item to run CodeWright's APIs on the fly. Type in the function name and any necessary parameters, and press . Refer to CodeWright's online help for information on CodeWright APIs. In addition to the menu item, the **API Command** prompt can be accessed with a special keystroke, appropriately named the Command Key. The API Command Key/Prompt is described next.


API Command Key

If you find you want to do something in CodeWright for which there is neither a key command nor a menu entry, you can probably do it using the Command Key:

- 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 , which in BRIEF also provides access to the BRIEF interpreter.
- In keymaps where there is no KeyCommand assignment, selecting **API Command** from the **Tools** menu will always access the prompt.

When you press the **Command Key** or choose **API Command** from the **Tools** menu, a prompt appears on the status line or in a popup dialog box, depending on which is enabled, like so:


Command:

You respond to the prompt by entering a CodeWright API function call, much as you would enter it in CodeWright's C source code. Press  to send the command off for processing.

Note: To specify a status line Command Key, choose the **Use Command Line Prompt** option on the **Customize | Environment | General** dialog. Refer to the topic *Command Line Response Editing*, in the online help, for command line key sequences.


Command Completion

The **API Command** prompt has a handy command completion feature which 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 popup dialog.

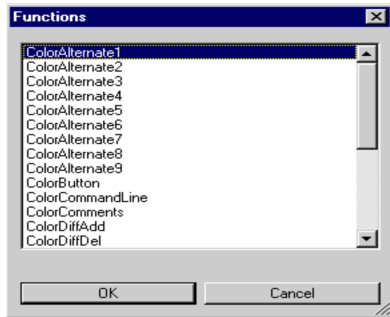
With command completion you can enter as much of a command (function name) as you can recall and press the  key to receive a list of all commands that begin with the same characters as the incomplete command.

API Command Completion Example


Complete the following steps to see an example of API command completion:

1. Select **Tools | API Command**.
2. Enter the word `color` and press . The following dialog displays.

Command Completion for "color"



3. 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  key without entering anything at the prompt. To filter the list more selectively, use the characters `.*` before and after characters that you know you want to see. For example, if you want to see all commands that contain the word 'tab' type the following in the Command Key:

`.*tab.*`

Then press .

(More information about using CodeWright's API and the Command Key can be found in the chapter *Extend CodeWright*.)

Customize|Libraries: Loading CodeWright Add-Ons

The **Libraries** option on the **Customize** menu is used for interactively loading CodeWright DLLs (or Add-Ons) that extend the functionality of the CodeWright program. A number of commonly used Add-Ons are contained in the list of **CodeWright Libraries** in the **Libraries** dialog. The **CodeWright Libraries** can be quickly and conveniently loaded by placing checkmarks next to the ones to be loaded. (More Add-Ons are available in the SUPPORT directory of the CodeWright CD or from Starbase's WebPages at <http://www.starbase.com>.)

Note: DLLs (Dynamic Link Libraries) are software used by Microsoft's Windows to provide services to applications. Some DLLs are automatically loaded when needed by a program, and 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 CodeWright 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 (described in the chapter on *Configuration Files & Command Line Parameters*) and loads the DLL into CodeWright:

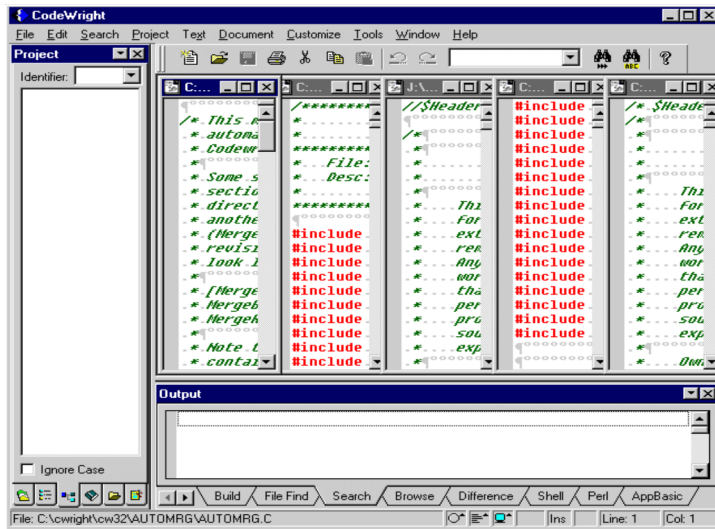
```
[LibPreload]  
LibPreload=CWHTML.DLL
```

General Environment Settings

The **General** tab of the **Environment** dialog (accessed from the **Customize** menu) is another item for which some early description is necessary. The **Environment** option accesses a dialog that deals with a number of different areas in CodeWright. For example, the **Environment** dialog is where backup files are specified, and it is also where CodeWright menus can be reconfigured. The other tabs of the **Environment** dialog are described in various chapters of this manual. The **Environment** dialog's **General** tab is given some attention here because it has options that broadly affect CodeWright's general behavior.

The **General** tab of the **Environment** dialog is used for setting various items in the CodeWright environment. The settings in this dialog affect file loading, file saving, file validation, keystroke and mouse behavior, and window options. For example, the option **Tile using single row/column** in the **General** tab affects the **Vertical/Horizontal Tile** options on the **Window** menu. When the option is marked, CodeWright has 'true' vertical and horizontal tiling, as displayed.

True Vertical Window Tiling



It is a good idea to set options in **Customize | Environment | General** before beginning any editing tasks.

Other concerns addressed by the settings in the **Environment** dialog include the following:

- ✓ Whether system prompts (e.g. **API Command** dialog/prompt) should popup 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**.
- ✓ Whether 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**.
- ✓ Whether a list of recently loaded files should appear at the bottom of the **File** menu. This option that can be quite handy for reloading files, and is turned on by marking **Show File List on File Menu**.
- ✓ Various file loading and file validation concerns.

The State tab allows you to save state information between CodeWright sessions. Ask yourself a couple of questions:

- ✓ Do you want CodeWright to remember and load the last file you were working on, all files, or no files?
- ✓ Do you want to save bookmarks between sessions?

API Command, Customize|Libraries, and Customize|Environment |General are important items that are referred to frequently throughout this manual. They have therefore been briefly covered as close to the beginning of the manual as possible.

Chapter 5

5- View Setups and Language Support

One of the first things you will likely want to do before beginning any editing in CodeWright is personalize the basics, like colors and scroll bars. In CodeWright, all colors, fonts, line numbers, scroll bars, and other window attributes are stored in View Setups. View Setups are "color schemes" with a kick. They store named sets of colors with individual fonts, rulers and other window-related features. Background and foreground colors can be controlled for individual screen elements, or for the window as a whole. The first part of this chapter talks about view setups and the dialog used for setting them.

The second part of this chapter covers CodeWright's main claim to fame, which is programming language support. Since colors are a primary concern in language support, and colors are a principal part of view setups, view setups seem to be a natural lead-in to language support. Language support features like ChromaCoding (lexical coloring for keywords, comments, etc), template expansion, and smart indenting are essential when editing source code. If those features are not immediately apparent during editing, it will be necessary to understand the process for turning them on or creating them.

View Setups: Colors, Window Attributes, Scrollbars, Fonts, Etc.

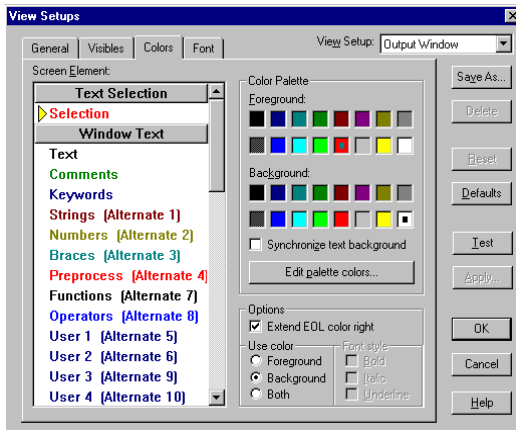
All items that pertain to CodeWright's visual environment, such as colors, scroll bars, line numbers, and window settings, are part of CodeWright view setups. View setups are controlled and manipulated in the **View Setups** dialog, which is accessed by clicking on the **Customize|View Setups** menu options. There are four tabs on the **View Setups** dialog: **General**, **Visibles**, **Font** and **Colors**.

- Use the **General** tab to set window attributes like scroll bars (with scrollbars come 'notches' for window-splitting functionality), 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.
- Use the **Visibles** tab to manipulate "Visibles", a term used to describe the characters that can be used to make invisible characters, such as spaces and tabs, appear on the display. To turn Visibles on, mark the **Visibles** option. To give the **Visibles** a slightly softer color, mark **Use Visibles Color**. The main listbox lists the characters that are currently being used for display. The characters can be changed using the combo-box just above the list of visibles.
- Use the **Font** tab to change screen fonts only. The fonts selected in the dialog do not affect printed documents. The print dialog has separate font settings for that. Items to note:
 - ✓ Font styles can be combined.
 - ✓ The font style for individual screen elements can be changed, but only when the selected font supports the styles.

Note: Many of the view setups, including *Default*, are set to use the 'Fixedsys' font. This gives two benefits: ChromaCoding font styles now display when first installed, and users that need the extended umlaut (European) characters are supported by default.
- Use the **Colors** tab to change screen colors. Colors can be changed for all text colors (i.e. the background as a whole), using the **Synchronize text background** option, or for individual screen elements only, by selecting the element and then changing the colors.

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

View Setups: Colors



Any combination of changes (whether the changes are color and font, window and color, Visibles and color, etc) made in the **View Setups** dialog can be saved with a name using the **Save As** button. Just click the **Save As** button, and give the view setup a new or existing name. Different 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 CodeWright 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.

CodeWright View Setups

CodeWright 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.

Note: Screen elements are the elements on the CodeWright 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 **View Setups** dialog.

- If the *Default* view setup is selected, additional screen elements are 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 chosen, additional screen elements 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 CodeWright windows. Changes made to this view setup will affect all windows that do not have view setups otherwise associated with them. If you would like to use another view setup as the default, there are two ways to do this:

- Use the **Save As** button in the **View Setups** dialog to save the selected view setup as **Default**, or
- 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 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

"Output Window *n*"

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

Using View Setups

As mentioned, windows in CodeWright are initially associated with the **Default** view setup. To use any other view setup, it must be associated with a CodeWright window. With this in mind, consider or do 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 current (on top) 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 windows (open or not) containing files of a certain file type (i.e. .C, .H, .CPP, etc), by choosing the view setup for the selected file type in **Customize | Language | Coloring**.

Since much of CodeWright's language support involves colors, and view setups have something to do with color, CodeWright's language support is described next.

CodeWright Language Support

CodeWright's language support can consist of the following elements:

- ChromaCoding
- Template Expansion
- Smart Indenting
- Brace Expansion

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 support Add-Ons are available but not loaded; in those cases, a simple procedure for loading the Add-Ons is all that is required to enable the support. Ultimately, support can be made for any programming language, through the creation of ChromaCoding Lexers and language Add-Ons. The following topics discuss lexers and Add-Ons, and the various ways they are configured to provide support for languages that are not immediately supported by CodeWright out-of-the-box.

Language Support Lexers and DLL's

The first thing any programmer expects to see when loading source code files into CodeWright is color, or ChromaCoding, for the syntax elements of the source code's language. A programmer might also expect other language support features, such as template expansion, to be immediately available. Depending on the language, however, this may not always be the case.

Some languages are supported by default in CodeWright. For example, when files with .C extensions are opened in CodeWright, 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 CodeWright out-of-the-box. When a language is supported out-of-the-box it often gets its functionality from CodeWright's CWSTART.DLL, which loads automatically each time CodeWright is launched.

For language support that is available but not loaded out-of-the-box, there are two options:

- The support can be obtained from external Add-Ons (DLLs) which must be loaded manually,

OR

- The support can be obtained from a **Lexer**, that can be created and/or modified in the **ChromaCoding Lexer Settings** dialog.

The topics that follow describe the two methods for adding language support to CodeWright. CodeWright language DLLs are covered first, followed by a discussion of CodeWright ChromaCoding Lexers. When reading the following information, keep in mind that ChromaCoding Lexers are the easiest way to create *new* language support for CodeWright. ChromaCoding provided by lexers is also faster, meaning that color display will be smoother. However, lexers do not support Smart Indenting, Template Expansion, or Brace Expansion. These features are provided by other means.

It is possible to use Add-Ons together with lexers. In such cases, the lexer's ChromaCoding functionality will take precedence, but the Add-On will still provide other language support features.

Language DLLs

When language support is obtained from a DLL, the first thing to do is make sure that the DLL for the language is in the CodeWright directory. Some language support DLLs are already in the CodeWright directory. Many more can be found in the \Support directory on CodeWright CDs, or can be downloaded from Starbase's webpages. On the webpages, the support comes in the form of a zip file, which contains the supporting DLL, some source code, perhaps a keyword file and a README file. CodeWright CDs maintain the same Add-On files in unzipped format, in the \Support directory.

Customize | Libraries

After making sure that the appropriate DLL is in the CodeWright directory, the library needs to be loaded. CodeWright libraries (including language support) are loaded using CodeWright's **Customize | Libraries** dialog. The DLLs will add functionality to CodeWright, whether that functionality is language support, or some other feature.

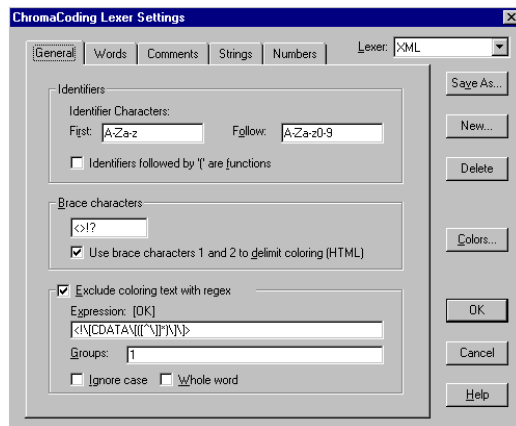
Some Add-Ons are listed in the **Libraries** dialog, in the list of **CodeWright Libraries**. If a language is not supported out-of-the-box, the first place to look is in the list of **CodeWright Libraries**. Loading a library from the list is as simple as putting a checkmark next to the item to be loaded. If the language is not listed explicitly, it may be available in the \Support directory of the CodeWright CD, as mentioned. The library can be loaded by clicking **Add** and browsing for the appropriate DLL. When loading an Add-On from the CD, it is a good idea to first copy the Add-On to a directory on your local hard drive (preferably the CodeWright installation directory) so that it will be available after the CD has been removed. For more information about the Libraries dialog, see the topic *Customize|Libraries: Loading CodeWright Add-Ons* in the chapter on *Command Key, Libraries and General Environment*.

ChromaCoding Lexers

ChromaCoding Lexers provide language support (i.e. ChromaCoding) for programming languages being edited in CodeWright. They store collections of settings to color various elements of the programming language's "vocabulary". Those "vocabulary" elements include identifiers, braces, preprocessors, keywords, operators, strings and comments.

Lexers are the easiest way to add support for a language that CodeWright did not previously support. ChromaCoding Lexers can be created or modified in the **ChromaCoding Lexer Settings** dialog.

ChromaCoding Lexer Settings



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

- Click the **ChromaCoding Lexers** item on the **Customize** menu.
- Click the **Settings** button on the **Customize|Language|Coloring** dialog.

Some lexers are provided in the drop-down list under the **ChromaCoding Lexer Settings** dialog's lexer combo box. Default settings on the tabs of the **ChromaCoding Lexer Settings** dialog will vary depending upon the lexer selected.

If a desired lexer does not exist, it can be created.

Creating a Lexer

To create a Lexer, complete the following sections:

- *Make a Lexer Name.*
- *Define Identifier/Brace Characters and Text to Exclude from Coloring.*
- *Define Keywords.*
- *Define Comments.*
- *Define String Delimiters.*
- *Define Numbers.*

Make a Lexer Name

To make a Lexer Name:

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

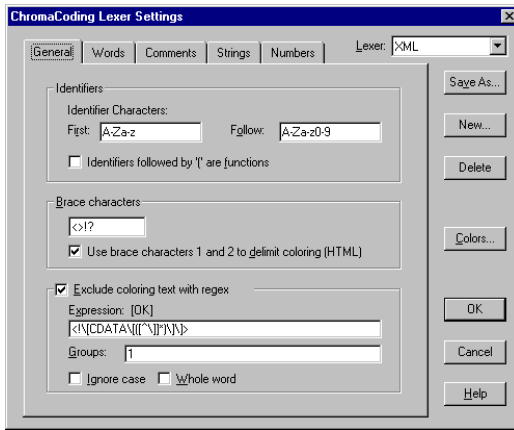
A new Lexer will appear in the **Lexer:** combo box.

Define Identifier/Brace Characters and Text to Exclude from Coloring

On the General tab of the **ChromaCoding Lexer Settings** dialog, define identifier characters and brace characters for the language, and any text to be excluded from coloring by this lexer.

After the lexer name has been created, identifier characters must be defined. Identifier characters are characters that distinguish "identifiers" from other elements of the language. Identifiers can best be described as a set of characters used to 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.

ChromaCoding Lexer Settings | General



Character classes (described under the topic *Regular Expressions* in the chapter on *Search and Replace and Navigational Tools*) can be used when defining identifier characters.

Example: `A-Za-z$`

`A-Za-z0-9$`

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

There are three fields in the **Identifier Characters** section:

- The **First** field is for characters that will normally appear as the first character of an Identifier.
- The **Follow** field is for characters that will normally follow the first character of each Identifier.
- A checkbox that indicates whether Identifiers followed by parentheses are functions (like C++).

The options in the **Brace Characters** section apply to certain languages only and indicate the following:

- If braces are used, which ones should be colored.
- Whether the first two braces defined should delimit keywords, numbers, and strings for coloring (i.e. elements are only colored when surrounded by braces, like HTML).

Mark the **Exclude Coloring Text with Regex** option to allow text that matches the regular expression you enter to be skipped over by other coloring definitions. Define the regex expression with the following fields:

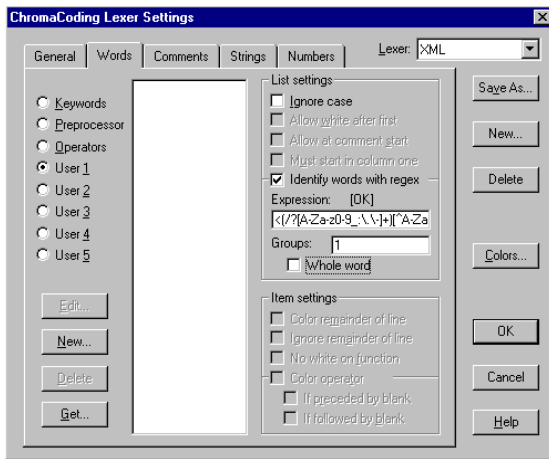
- **Expression:** The regular **Expression** pattern is used to parse matches from files that use the corresponding ChromaCoding lexer. The lexer will then exclude from coloring the parts of the matches designated by the reference groups within the pattern.
 - ✓ The word [illegal] will appear above the edit box if the pattern in the box is not a valid regular expression. The word [OK] will appear if the regular expression is valid.
 - ✓ It might be helpful to create the regular expression in the **Search** dialog so you can readily see what is matched. Once the pattern has been created, it can be pasted in the **Expression** box.
- **Groups:** Place one or more space-separated numbers in the **Groups** box to indicate which reference groups in the **Expression** box should be used to match strings that are to be excluded from coloring.

If no reference groups are specified, the whole expression in the **Expression** box will be used. If more than one group exists in the pattern, only the groups that are specified by the numbers in the **Groups** box will be used. Note that reference groups in regular expression patterns are counted from left to right starting at 1.
- **Ignore Case:** Mark **Ignore Case** to have any occurrence of a match by the **Expression** pattern excluded from coloring, regardless of case (upper, lower or mixed).
- **Whole Word:** Mark **Whole word** to have the ChromaCoding lexer exclude from coloring only matches that are surrounded by whitespace or Word Delimiters.

Define Keywords

On the **Words** tab of the **ChromaCoding Lexer Settings** dialog, add any keywords, operators (e.g. +,-,*,&), or preprocessor words (if appropriate) that the ChromaCoding Lexer should color.

ChromaCoding Lexer Settings | Words



Use the following steps to add words:

1. Choose the radio button for the group (i.e. **Keywords**, **Operators**, etc) that the items will be added to.
2. Click the **New** button on the left.
3. In the resulting **Add Words** dialog, type the appropriate character(s) for the word to add.

Notes:

- ✓ Items can be deleted from a list by selecting the item and pressing **Delete**.
- ✓ The **Get** button adds items from a file. Click it to bring up the **Get Words from File** dialog. Use the **Get Words from File** dialog to browse for the file containing the items to be added. The **Get Words from File** dialog can only read whitespace separated words. It can not differentiate words from other characters like parentheses, colons, etc.
- ✓ 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: Colors, Window Attributes, Scrollbars, Fonts, etc.*)

- ✓ The option **Identify words with regex** in **ChromaCoding Lexer Settings|Words** is for file types (languages) that allow keywords to be recognized that may be specific to a document, but are not part of the language definition.

Since these dynamic keywords may vary from file to file, there is no way to define a list of them for the ChromaCoding lexer. Because of this, the lexer will not be able to color them in CodeWright.

Identify words with regex solves this problem. It enables controls and an edit box that are used to define and configure a regular expression; the regular expression parses words from files that use the corresponding ChromaCoding lexer. The lexer will then know which words to color.

It might be helpful to create the regular expression in the **Search** dialog so you can readily see what is matched. Once the pattern has been created, it can be pasted in the **Expression** box.

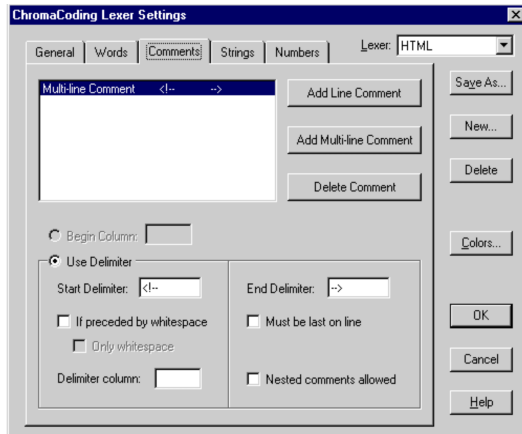
- ✓ The **List Settings** section has items that apply to whole lists of keywords (i.e. if **keywords** is chosen, the options apply to the list of keyword elements, if **operators** is chosen, the options apply to the list of operator elements). Certain items apply only to certain lists. For example, **Allow White After First** (which causes preprocessors to color even if whitespace appears after the first character) can be used for lists of Preprocessors only.
- ✓ The **Item Settings** section has options that only apply to individually selected items.
- ✓ Certain checkboxes 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 handy for languages like C++ that have preprocessor lines like:
 - `#include "exports.h"`
 - where "exports.h" would otherwise be incorrectly colored as a string.

(Refer to CodeWright's online help for additional information on the options available on the **Words** tab.)

Define Comments

On the **Comments** tab, define all the appropriate comment delimiters for each comment set. The configurations made in this dialog will tell the ChromaCoding Lexer when to color comments.

ChromaCoding Lexer Settings | Comments



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.

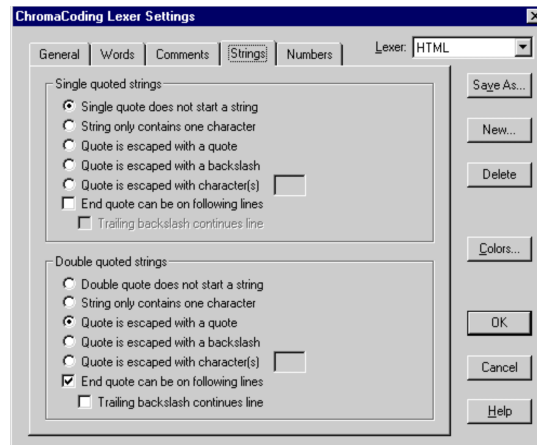
Finish setting up the comment support by completing the fields and command options that apply to the characteristics of the comments for the language in question.

(See CodeWright's online help for more information on the **Comments** tab.)

Define String Delimiters

On the **Strings** tab of the **ChromaCoding Lexer Settings** dialog, 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) that specify that **Single/Double quote does not start a string**.

ChromaCoding Lexer Settings | Strings



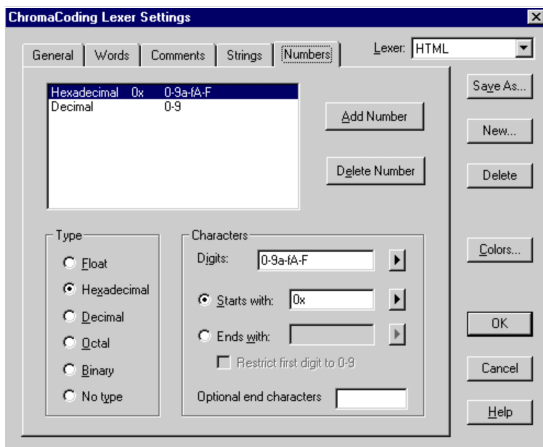
Define Numbers

On the **Numbers** tab, define any numbers that should be colored by the ChromaCoding Lexer for the language in question. The available number types are:

- Float
- 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.

ChromaCoding Lexer Settings | Numbers



Use the following steps to define number elements:

1. Click **Add Number**, then choose the number type to be applied. A new item will appear in the list box at the top of the dialog.
2. 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 quickly inserted using the popup menu displayed with the black right-arrow button , to the immediate right of the **Digits** field.

- ✓ 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. This is for programming languages that delimit numbers with specific starting or ending characters.

The **Starts With** and **Ends With** fields also have black right-arrow buttons to access popup 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 specified characters *only* if the first digit is numerical.
- ✓ The field labeled **Optional end characters** specifies that ChromaCoding include the characters in the box when coloring any numbers that end with them.

Completing the Lexer

- When all 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, and click the **Delete** button.

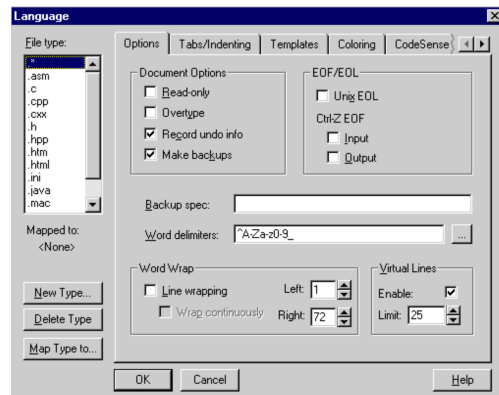
Click **Cancel** to undo any changes made during the current session of the **ChromaCoding Lexer Settings** dialog.

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 items.

The **Language** dialog is where all CodeWright's language features are stored and controlled. CodeWright's language features consist of settings and support that are associated with file type extensions (i.e. .C, .TXT, .ASM, etc). The available features vary depending on the extension of the file.

Customize|Language



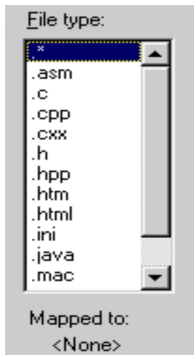
Some of the options in the **Language** dialog are available for any file type, while others are only available for file types for which a language support module is available. For example, the **Paste Indenting** feature can be used for any file type, while **Brace Expansion** features (described in the chapter on *Editing & Printing*) are only available for C and C++ files. 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 will be described here, along with a few other elements of the dialog. The main thing to remember when making changes in the **Language** dialog is that the changes are file-type specific. That is, changes apply only to files with extensions that are 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 CodeWright.

File Types



A file type is the extension of a given file (e.g. **.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 in the list. If it is not in the list, it can be added, or the default settings (indicated by file type **.***) will apply. To select more than one extension at a time, use **CTRL**- or **SHIFT**-click.

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.
- **UNIX EOL** controls whether UNIX EOL characters should be inserted when the Enter key is pressed in files of the selected type.
- Whether and how **Word Wrap** should be used 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 CodeWright 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 turning on, examining, deleting, and redefining language templates associated with the selected extension. More information about templates is available in the chapter on *Editing & Printing*.

Coloring Tab

The **Coloring** tab of the **Language** dialog has options for controlling how ChromaCoding operates. It is also where CodeWright 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 CodeWright. 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 CodeWright 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.

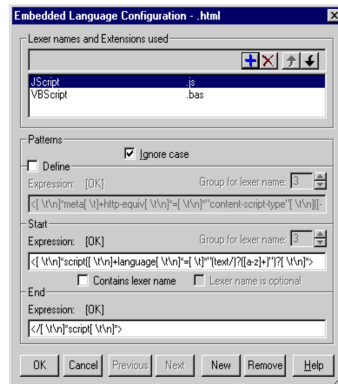
These buttons are not available when the **Lexer** option is chosen.

- 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, CodeWright will be able to distinguish the embedded language from the main language, and provide support for both.

Embedded Language Configuration



Help for the dialog is available online, under the topic *Embedded Language Configuration*. See the topic *Regular Expressions* in the chapter *Search and Replace and 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 Name Completion, Outline Symbols, and CodeSense features. These features parse specified text from files being edited which is then used for editing or navigational purposes. More information about Name Completion and CodeSense can be found in the chapter on *Editing & Printing*. Symbols are described in the chapter on *Search and Replace and 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 more detail in the chapter on *Checking and Reformatting Files*.

Comments Tab

The **Comments** tab of the **Language** dialog is 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 the chapter on *Editing & Printing*.

Adding a New File Type to the Language Dialog

Before CodeWright can color a file, it has to know the file's type (i.e. .C, .TXT, etc). Usually the file type will already be defined, but not always. If it isn't defined, it will need to be added to the **File Type** list in the **Language** dialog. Do the following to add a new file type and then associate it with language support:

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 the chapter on *Editing & Printing*) 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 that the corresponding Add-On or lexer has been designed to support should also be available.

Aliasing

If there isn't a language support DLL for a particular file type, there are two options available: add support by creating a lexer (described in the previous section) or create a DLL. There is some information on creating a language DLL at the end of this chapter and also in the chapter on *Extend CodeWright*.

An alternative way to add language support is to alias the unsupported file type with a supported one. If the code style of the unsupported language is similar to one that is supported, all the templates, keywords and coloring for the existing file type will be 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:

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 will use the language support of the file type they are mapped to.

Some of the tabs in the **Language** dialog will be unavailable for mapped or aliased files. This is because mapped file types use the language-specific functions of the type to which they are mapped. In such cases, mapped file types can only be changed by changing options for the original file type to which the unsupported language is mapped.

Creating a Language Support DLL

If there is no Add-On available to support the programming language you are editing, it is possible to make one. A module is provided in the \Support directory of the CodeWright CD that contains source code for a generic language support Add-On. The source code could be used as a frame of reference for building a custom DLL that will provide support for the language in question. The module is called GENICL50.

An alternative option when creating a language Add-On is to use the source code for the language currently supported that most closely matches the language to be added. Source code for most CodeWright DLLs (including CWSTARTDLL) is available in subdirectories of the main CodeWright directory for **Full** CodeWright installations. The closer the commenting of the supported language is to the language for which support is being added, the simpler the task will be. (Creation of a language support DLL is an extensive topic for which more detailed information is provided in the chapter on *Extend CodeWright*.)

Chapter 6

6- Editing & Printing

This chapter covers most of CodeWright's special features that help make editing easier. Things like Template and Brace Expansion will be covered extensively. Functions and dialogs used for inserting Comment Boxes, File and Function Headers, CodeFolio Snippets and completed API functions (API Assistant and CodeSense) will also be discussed along with "hex-mode" editing and CodeWright's COBOL, HTML and XML features. The chapter also describes CodeWright's **Print** dialog and explains the process for configuring CodeWright's online help to access help from different environments.

Templates and Brace Expansion

Refer to the following discussion of Templates, Template Macros, CodeFolio Snippets, and Brace Matching and Expansion.

Templates

Templates may be language specific (i.e. associated with a particular programming language) or language independent. They are useful for taking the drudgery out of repetitive editing tasks. They are designed to insert frequently used text into files at the press of a button, or to automatically expand common statements used in programming languages when appropriate abbreviations are typed.

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 in the following way (using C++ "if" statement as an example):

Example: Type if, and then press the space bar.

The appropriate ending statement and any necessary parentheses, curly braces, etc, will be inserted in the appropriate positions for the statement, like so:

```
if ( )
```

CodeWright's templates have a macro capability that makes them useful for much more than just language constructs. The next sections go over the necessary information for creating templates used for language constructs and they describe the advanced features provided by template macros.

Creating and Modifying Language Specific Templates

Before adding or changing a template, it is a good idea to become familiar with the templates that are already defined. CodeWright associates templates with the file type of any given programming language in order to make them language specific.

Two methods are provided for adding template constructs, or changing existing ones:

- Templates defined for a particular file type can be viewed, added and changed on the **Customize | Language | Templates** dialog . Use the dialog to see, create and modify the abbreviations that trigger template expansion and the string values associated with them.
- Add templates by directly editing CodeWright's configuration file. Use a call to **ExtAssignTemplate** under the [Templates] heading of the CWRIGHT.INI file. **ExtAssignTemplate** is a CodeWright API.

There are three string parameters used by **ExtAssignTemplate**. The first tells what file extension to associate the template with (e.g., .C, .SC...). The second specifies a word or abbreviation that is to trigger the template expansion, for example "if", "else", and so on. The third string is the template itself.

For more information:

- ✓ Help for CodeWright APIs can be found in CodeWright's online help using the **Search For Help On...** menu item on the **Help** menu.
- ✓ More information on CWRIGHT.INI is provided in the chapter on *Configuration Files & Command Line Parameters*.

There are special characters defined for use in language templates that give CodeWright useful instructions. The following table defines those characters:

Character	Purpose
\n	New Line. Simulates pressing enter at this point.
&	Specifies cursor position after template insertion.
@	Issues a backspace.
\c	Insert 'c' literally, (e.g., \&, \@, \\)
\t	Insert a tab.

Examples:

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

Here are some sample template strings, that give an idea of how these special characters are used to construct templates:

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

Finally, here is how the complete **ExtAssignTemplate** line might look in the configuration file:

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

This example adds a template for use when editing files with the extension .PAS. When the word "proc" is typed, 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.

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

The templates described up to this point are language specific. Templates that can be used independently of a language's file type extension will be described next. Template macros (macros that perform specific functions inside templates) will also be described. Note that template macros can be used in language specific templates as well as non-language-specific templates.

Two templates provided with CodeWright 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, or new ones can be created, that will insert any kind of text, whether for a programming language or not. The templates are contained in two files in the CodeWright home directory named 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 CodeWright button or Keystroke. Details for the **ExtExpandTemplate** function can be found in CodeWright's online help.

Note: The **Edit** toolbar is turned on by marking the **Visible** option for **Edit** in the list of toolbars on the **Toolbars** tab of the **Customize | Toolbars** dialog.

To see examples of how **ExtExpandTemplate** is used, take a look at the function bindings for the function and file header buttons on the **Edit** toolbar. The function bindings for the toolbars can be viewed on the **Bindings** tab of the **Customize | 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. As you scroll through the buttons, you will see the function binding for each button in the **Function Bindings** edit box at the bottom of the dialog. The first two buttons on the **Edit** toolbar are the buttons that insert the headers. The function binding for the function header appears like this:

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

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

The function binding for the file header appears like this:

```
ExtExpandTemplate %ffile.tpl$
```

Note that the two function bindings are exactly the same except for the names of the template files. Template macros are described next.

Template Macros

True to their name, template macros like `%f` can also be used within templates. FUNCT.TPL contains the `%q` macro. When it is expanded, it presents a prompt that queries the user for information, which will be used when the template is inserted. In the following examples we show before and after shots of FUNCT.TPL after it has been modified with extra template macros, to illustrate their use.

Example: 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.

Example: FUNCT.TPL with modifications:

```
/%rep*60
** %qEnter function name:$
*
*   PARAMETERS:  &
*
*   DESCRIPTION:
*
*   RETURNS:
*
*   CREATED:  %date  %time
*
*   BY:  %eUSERNAME$
*/
```

These are the macros that were added:

<code>%rep</code>	This macro repeats the* character 60 times.
<code>&</code>	This specifies where the cursor should be placed at the end of the insertion.
<code>%date</code>	This inserts the current date.
<code>%time</code>	This inserts the current time.
<code>%e</code>	This inserts the value associated with the specified environment variable, USERNAME. Again, the \$ character is used to delimit the string.


Below is an example of a header that was added by the expansion of the modified template file:

```

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

These are just a few of the things that can be done with template macros. A table containing a complete list of the % template macros available follows:

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 date string at cursor position. (mm/dd/yy format)
<code>%db</code>	Delete to the beginning of the line.
<code>%dcNum</code>	Delete <i>Num</i> characters at cursor.

Macros in Templates	
% Macro form	Description
%de	Delete to the end of the line.
%dl <i>Num</i>	Delete <i>Num</i> lines, beginning with the current.
%dw	Delete the word at the cursor.
%e <i>EnVar</i> \$	Insert the string value associated with environment variable <i>EnVar</i> .
%f <i>Filename</i> \$	Insert the named file at the cursor. Any template macros within the file are also processed.
%home	Move the cursor to the beginning of the line.
%line <i>Num</i>	Move the cursor to the line named by <i>Num</i> .
%md <i>Num</i>	Move down <i>Num</i> lines.
%meof	Move the cursor to the end of the file.
%meol	Move the cursor to the end of the line.
%ml <i>Num</i>	Move the cursor left by <i>Num</i> columns.
%mr <i>Num</i>	Move the cursor right by <i>Num</i> columns.
%mu <i>Num</i>	Move up <i>Num</i> lines.
% <i>Num</i>	<p>When <i>Num</i> is 0 to 9, it 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 CWRIGHTEXT. The macros 0 through 3 are used for custom indentation in predefined language templates. See ExtSetTemplateMacro.</p> <p>Higher numbered macros (10 through 31) are not extension specific, but are otherwise similarly definable. They are reserved for the use of individual users.</p>
%open	Open a new line following the current. Similar to going to the end of the line and pressing  .
%q <i>Prompt</i> \$	Query for string to insert, using the string <i>Prompt</i> to prompt the user.
%rep <i>CNum</i>	Insert <i>Num</i> repetitions of character <i>C</i> .

Macros in Templates	
% Macro form	Description
%response	Insert data acquired from %q macro. Allows the data to be used more than once. More than one response can be used by adding a number to the end of the macro, e.g. %response1.
%restore	Restore a saved position.
%save	Save a position for later restoration.
%time	Insert a formatted time string. (hh:mm:ss)
%tof	Move cursor to the top (first line) of the file. Requires %home to position cursor at the first character of the file.
%xFuncCall\$	Execute the CodeWright API function call in <i>FuncCall</i> . This may be any function that could be assigned to a key or otherwise executed through LibFunctionExec .

The function and file headers and template macros described in this section can also be used and inserted with CodeWright's CodeFolio Snippets feature, described in the next section *CodeFolio Snippets*.

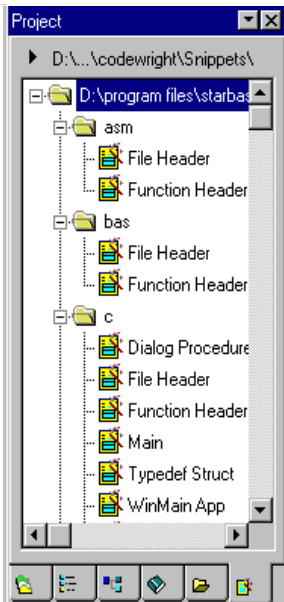
CodeFolio Snippets

CodeFolio Snippets are similar to templates, but more flexible in how they are used. They are pre-defined chunks of code or text, which may be inserted into your current document. The style/structure of code or text following a Snippet is 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 CodeWright, click on the **CodeFolio** (right-most) tab of the Project Window. The directory tree is organized by language type, as follows:

Project Window: Code Folio Tab



In addition to using the Snippets that are shipped with CodeWright, you may add directories containing your own Snippets to the **CodeFolio** tab (refer to *Adding or Removing Snippets Directories*, in this chapter). Snippets may 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. Begin by creating a new document (**File|New**) or opening an existing document (**File|Open**) into the Client Area. Then click on the **CodeFolio** (right-most) tab of the Project Window:



Execute a Snippet by doing 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 popup menu. The Snippet will be inserted at your cursor position in the file.

Below is an example of the CFUNCT.TPL snippet after it has been executed in a new document:

C Function Header Snippet

```
/*
 * FUNCTION: Test_Function
 *
 * PARAMETERS:
 *
 * DESCRIPTION:
 *
 * RETURNS:
 *
 */
```

Other examples of Snippets include the following:

HTML Document Setup

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

VBFUNCTION.TPL

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

Adding a Code Snippet

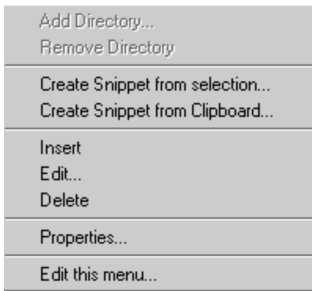
You may save portions of code that you have written as Snippets for later use. Complete the following steps:

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

Creating a Snippet from the Current Document or Clipboard

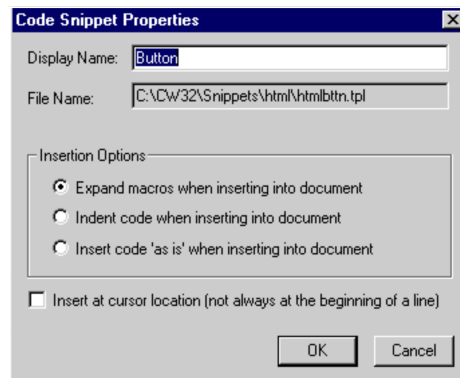
You can paste selected text from the current document, or text from the clipboard, and save that text as a new Snippet. Complete the following steps:

1. Copy the desired text to the clipboard, or select the text in your document.
2. Right-click on the Snippet directory in which 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. Complete the **Code Snippet Properties** dialog that displays, defining the display name and storage directory for your new snippet. You can also choose whether to expand macros (see the topic *Template Macros* for information about macros) upon insertion, whether to match the level of indentation at the insertion location, or whether to insert code “as is”. If you want to insert the snippet at the cursor (even if mid-line) mark the checkbox accordingly.

Code Snippet Properties



Deleting or Renaming a Snippet

You can delete or change the display name of a Snippet from the popup menu.

- To permanently delete a Snippet from your system:
 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:
 1. Right-click on the name of the Snippet in the **CodeFolio** tab's tree directory.
 2. Select **Rename**. The name of the Snippet should now be highlighted.
 3. Edit the name directly.

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

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

Editing a Snippet

If you wish 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 Template** from the popup 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.

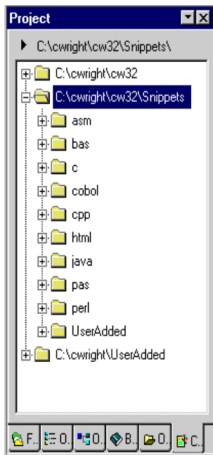
Note: To create or remove a subdirectory of an existing Snippets directory, you must add/delete the folder outside of the CodeFolio tab (e.g. from Microsoft Explorer).

To add a root level directory:

1. Right-click on the directory under which the new directory should be placed.
2. Select **Add Directory** from the popup menu.
3. Browse for and select the desired Snippets directory on your system.

A sample directory tree with a user-added directory named *UserAdded* follows:

CodeFolio Snippets with User-Added Directory



To remove a root level directory:

1. Right-click on the directory you wish to remove from view.
2. Select **Remove Directory** from the popup 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 CodeWright works several ways:

- 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.

Brace matching is performed by functions assigned to keys or buttons. Several of these functions are already available on buttons on CodeWright's **Edit** toolbar. To turn on the **Edit** toolbar, mark the **Visible** option for the **Edit** toolbar on the **Customize | Toolbars | Toolbars** dialog. You may wish to customize these assignments or make new assignments. Information on customizing buttons and keystrokes is found in the chapter *Custom Interface*.

Brace Function: Finding Unmatched Braces

There is a function supplied with CodeWright, **Brace()**, that looks for unmatched braces in the current buffer. All braces that are to be matched can be specified. One way to invoke the **Brace** function is to use CodeWright's API Command Key. The API Command Key accesses a prompt or dialog from which CodeWright 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 the chapter *Command Key, Libraries, & Environment*.

To match all curly braces within a file, respond to the **Command:** prompt as follows:

Command: Brace TRUE

- ✓ To ignore braces in comments, just omit the TRUE parameter, or supply FALSE as a parameter instead.

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**, which looks for the next left brace or parenthesis, locates its mate, and then highlights the text between them. On subsequent calls, this function will find pairs nested within the highlighted set, or a pair following the highlighted set.
- The second function is **BraceMatch**, which 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 neither a left nor a right parenthesis or brace is at the cursor position, the function searches forward for the next matching set.

Note: Since **BraceMatch** looks at the current cursor position, it is not effective for highlighting a series of blocks in sequence. Subsequent calls, will find 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 just positions the cursor at the corresponding object.

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

BraceFindEx

BraceFindEx() is another brace-matching function that can be used from CodeWright'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 is used follows:

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

```
BraceFindEx BRACE_BRACES | BRACE_PARENS | BRACE_FORWARD
```

Brace functions are documented in CodeWright's online help.

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 instantly inserts braces whenever the left brace ({) key is typed, while template expansion doesn't insert braces until the space bar is pressed. The sequence used by template expansion may be unnatural for some programmers, and therefore, brace expansion provides an alternative. Brace expansion may also be useful for users who do not want full template expansion, but who want the benefits of automatic brace insertion.

Align Beginning and End of Block

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

Example: If the alignment setting on the **Customize | Language | Tabs/Indenting** is set to **Unindented Block**, the closing block delimiter will be aligned with the opening block delimiter as it is typed. This will occur whether the closing delimiter is typed at column 3 or 30.

Align <begin/end block> is similar to **Brace Expansion**, except that it does not instantly insert both the closing and opening block delimiters as they are typed. It just realigns the closing block delimiter as it is 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

Refer to the following discussions on:

- *Setting Spaces and Tabs*
- *Seek Indentation and Smart Indenting*
- *Block 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, it is undesirable to have actual tab characters inserted in a file. CodeWright 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, mark **Use Spaces**. Remember that all changes made in the **Language** dialog are language specific, so be sure 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. CodeWright 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 4-column tab stop would be set by inserting the following numbers: 5 9.


It works this way:


The cursor starts in column 1. When the tab key is pressed, 4 columns are added, putting the cursor in the 5th column. When the tab key is pressed again, 4 columns are added, putting the cursor in the 9th 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

Two Indenting features, enabled in **Customize | Language | Tabs/Indenting**, are designed to 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. The white space options consist 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 .

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

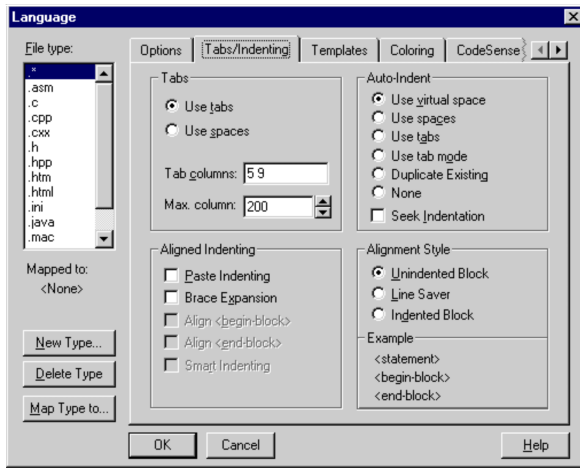
Smart Indenting may be thought of as an extension of the **Seek-Indentation** feature. **Smart Indenting** may be used with **Seek Indentation** turned off, but leaving it on ensures the most consistent appearance.

True to its name, **Smart Indenting** will be smart enough not to 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 have been discussed in this chapter so far deal with potentially indenting blocks of code. Since preferences may differ as to how indenting should occur, CodeWright offers several different indentation styles to choose from. These styles are set on the **Tabs/Indenting** tab of the **Language** dialog.

Customize | Language | Tabs/Indenting



The **Alignment Style** option in **Customize | Language | Tabs/Indenting** is for selecting between several popular forms of block indentation to be used for brace expansion and template expansion. Choose from the following indentation styles:

- **Unindented Alignment:** Places the block delimiters ({ and }, BEGIN and END, or whatever) at the same level of indent as the controlling statement that precedes the block. Statements between the block delimiters are indented.
- **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.
- **Indented Alignment:** Block delimiters ({ and }, BEGIN and END, for example) are indented, compared to the controlling statement that precedes the block. Statements between the block delimiters are not indented.

Thus far this chapter has discussed CodeWright features that aid with repetitious editing tasks. The chapter will continue along these lines by discussing features that help with editing by offering information and reminders about parts of the file being edited. These features include Name Completion, CodeSense, and Comments and Comment Boxes, as well as hex editing, HTML editing, Clipboard and Scrap features, the API Assistant, and help index file configuration.


Name Completion

CodeWright's Name Completion will automatically complete strings that occur in either the current document or the CodeSense database, as the strings are typed.

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**. 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).

Name Completion is turned on by marking **Name Completion** in **Customize | Language | CodeSense**. The option is enabled by default. Additional settings are also available on this dialog.

- ✓ To use Name Completion, type the first few characters of the variable, class, symbol name or other word to be completed and press -Space.

CodeSense

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

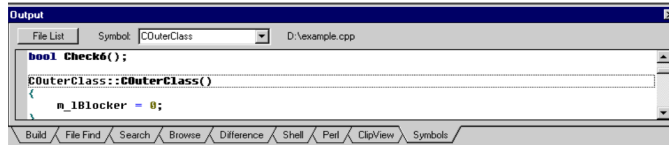
- From C/C++ and Java source files in any of the library databases listed in **Customize | CodeSense Global Configuration**.
- From C/C++ and Java source files in CodeWright projects.
- From C/C++ and Java source files that are listed in the file list on CodeWright's Symbols Window. Files in projects 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:

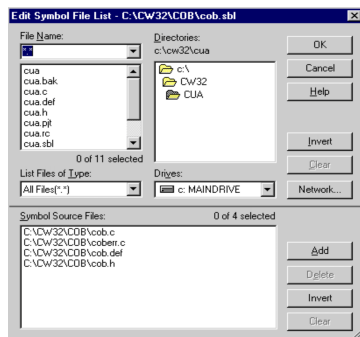
1. Click the **Symbols** tab of the Output Window (located by default at the bottom of the CodeWright screen).

Symbols Window




2. Click the **File List** button to access the **Edit Symbol File List** dialog.

Edit Symbol File List



3. Type in or browse for the file to be added to the list.

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

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

CodeSense for Files that are Open in CodeWright

In addition to the information that CodeSense gets from library and project databases, it also gets information from C/C++ and Java files that are open in CodeWright. 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 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.

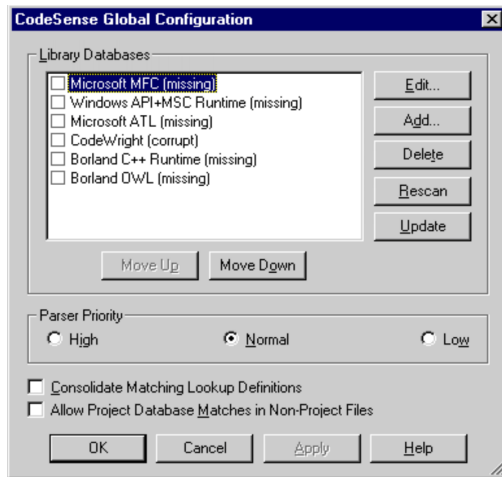
CodeSense Global Configuration Dialog

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++ and Java files. Several default library databases can be installed with the CodeWright installation. They will appear in the dialog if they were installed. If the appropriate library database is not available with the CodeWright 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 CodeWright 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.

CodeSense Global Configuration



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

- From the **Customize** menu, by clicking **CodeSense Global Configuration**.
- From **Customize | Language | CodeSense**, by pressing **Global Configuration**.
- From the CodeSense popup menu (right-click on  on the status bar), 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 use the **Browse** button to browse for the correct location.
4. 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 Java. All .JAVA files will be scanned regardless.

6. 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 CodeWright.

Edit CodeSense Library Database Location

The default library databases provided with the CodeWright 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 use the **Browse** button to browse for the correct location.
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 Java. All .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 CodeWright.

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**.

Note: Databases that were created from the files in a CodeWright project must be deleted outside of CodeWright, from Windows Explorer. CodeWright 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

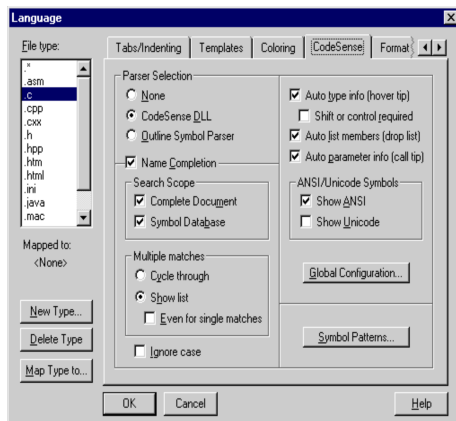
The CodeSense scanning process works on a background thread. It should not interfere with other CodeWright processes. It has been known, however, to occasionally be time and resource intensive. If you find that your CPU usage is high because of CodeSense, you may want to set the parser priority to low. 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:

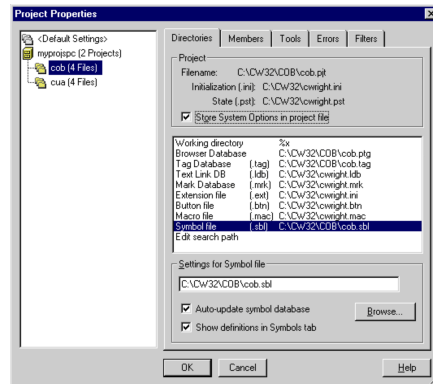
- **CodeSense DLL** is marked in **Customize | Language | CodeSense** (marked by default for .C, .CPP, .H, .HPP, and .JAVA files).

Customize | Language | CodeSense



- Auto-update symbol database and Show Definitions in Symbols tab are marked for the Symbol file option in Project|Properties|Directories.

Project|Properties|Directories



- A CodeWright project containing C/C++ and Java files is open and/or .C, .CPP, .H, .HPP, 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 CodeWright home directory.
- The .SBL file is located in the CodeWright home directory when no project is open and is called CWRIGHT.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 CodeWright 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 including CWRIGHT.SBL 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 CodeWright's Output Window.
2. Select **Rescan symbol DB files** from the resulting popup menu.



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

1. Exit CodeWright 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**.

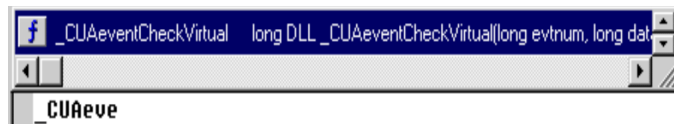
CodeSense: Main Features

CodeSense can be used once databases have been created. Before using it, though, it is necessary to choose options that control its behavior. Some CodeSense options are found in **Customize|Language|CodeSense** and some can be found in the **CodeSense Global Configuration** dialog. The main features of CodeSense are described next.

Name Completion

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

Example:




Click a selected symbol's representative image to access the symbol's definition.

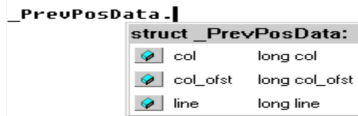
The following **Name Completion** options are available in the **CodeSense** dialog:

- **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



To have CodeSense automatically display member lists when appropriately following symbols such as structs, classes, unions, or enums with '.', '>', or '::', mark **Auto-list Members** in **Customize | Language | CodeSense**. The option is marked by default for .C, .CPP, .H, .HPP, and .JAVA file types. Once the list is displayed, use it by either selecting the desired symbol and pressing , or by simply clicking on the desired symbol. It will insert automatically.



Example:



Click a symbol's representative image to access the symbol's definition.

Auto-type Info

To have CodeSense display a Tooltip containing a symbol's definition as the mouse cursor hovers over it, mark **Auto-type info** in **Customize | Language | CodeSense**. The option is marked by default for .C, .CPP, .H, .HPP, and JAVA file types. Mark **Shift or control required** in **Customize | Language | CodeSense** to have the  or  keys control when the tooltip appears. The **Shift or control required** option works with **Auto-type Info** in the following way:

- If the **Auto-type Info** option is NOT marked, the **Shift or Control required** option is disabled, and tooltips are completely disabled.
- If the **Auto-type Info** option is marked, but **Shift or Control required** is not, tooltips ALWAYS appear when the mouse cursor hovers over a symbol.
- If the **Auto-type Info** and the **Shift or Control required** options are both marked, pressing  or  while the mouse hovers over a valid symbol causes the tooltip to appear when it otherwise would not.

Example:

```
EventRegister(EVENT_BUFFER_CREATED, EVENT_NORMAL, handler);  
 CW_DECLSPEC HEVENT WINAPI EventRegister(UINT, UINT, LPCTSTR);
```

Click a symbol's representative image to access the symbol's definition.

Auto-Parameter Info

To have CodeSense display a tooltip of appropriate parameters as a function is being typed, mark **Auto-Parameter Info** in **Customize|Language|CodeSense**. The tooltip displays when the function's left parenthesis is typed. As a parameter is typed, and commas are entered, successive parameters in the tooltip become bold, indicating that they are next in line to be entered.

- ✓ Bold type will not appear for systems that use a bold font for tooltips.

Example:

```
MsgConfirm(LPXSTR str,|
  CW_DECLSPEC int WINAPI MsgConfirm(LPXSTR, LPXSTR)
```

Click a selected symbol's representative icon to access the symbol's definition.

Extending CodeSense Functionality

Consider the following additional options for enhancing your use of the CodeSense features.

ANSI and Unicode CodeSense Translations

To have CodeSense specifically parse and display ANSI and/or Unicode versions of the Win32 API, mark **Show ANSI** and/or **Show Unicode** in **Customize|Language|CodeSense**. If these options are not marked, ANSI and Unicode versions of Win32 APIs will not be displayed. See CodeWright's online help topic *CodeSense Tab* for more information.

Disable CodeSense for Sections of Code Only

Two macros can be added to comments in C/C++ files to disable the CodeSense parser for specific sections of code. This can be handy for pieces of code that are not handled well by the parser.

The macros are:

- `TURN_CODESENSE_OFF`
- `TURN_CODESENSE_ON`

Example: The CodeSense parser will not parse the following piece of code:

```
//TURN_CODESENSE_OFF
printf("Don't parse this");
//TURN_CODESENSE_ON
```

Project Matches in Non-Project Files

To have CodeSense display project database matches even if the file being edited is not part of the current project, mark **Allow Project Database Matches in Non-Project Files** in **Customize | CodeSense Global Configuration**.

For example, consider that the current CodeWright project and the file being edited each contain a function named 'foo'. However, the file being edited does not belong to the current project. CodeSense has scanned both 'foo' functions and placed them in databases or in memory-- the project's function having been stored in the project-database, and the current buffer's function having been stored in memory. If Allow Project Database Matches in Non-Project Files is marked, both 'foo' functions will display in CodeSense drop-down lists. If it is not convenient to see matches for both functions, leave the option unmarked.

Consolidate Matching Lookup Definitions

To prevent CodeSense from displaying multiple matches for symbols that have the exact same definition but that come from different locations in the source files, mark **Consolidate matching Lookup Definitions** in **Customize | CodeSense Global Configuration**. If this option is marked, only one matching definition will display in the various CodeSense drop down lists, even if multiple matches are available in the corresponding CodeSense library database. Note that limiting matching definitions in drop-down lists will also limit the CodeSense 'Goto' functionality making only one matching symbol-definition available for access.

Symbol Lookups

CodeSense symbol lookups are done in a context-sensitive (scope sensitive) manner. The buffer offset of a symbol is used to determine the context. For example, in the following pseudo code, the scope of the first "str" would be "global, <foo>, <while block>, <if block>". Therefore the definition of "str" found in the else block would not qualify as the definition of the "str" in the if block.

```
void foo( void )
{
    while (TRUE)
    {
        if (TRUE)
        {
            printf( str );
        }
        else
        {
            LPSTR str = "foo";
        }
    }
}
```

The following additional information pertains to symbol lookups:

- When doing a member lookup, CodeSense does not care whether '::', '->', or '.' is typed. It will display the members regardless. For example, typing "CString->" will display the members of CString even though a compiler would expect "CString" to be followed by '::'.
- CodeSense won't display member lists for function return values (e.g. "GetWindow()->").
- Libraries and/or project files that are currently being scanned or parsed are not available for lookups.
- If a project database is being indexed or compacted, CodeSense will wait for a few seconds for it to finish. If it does not finish, the whole project database will be unavailable for lookups (i.e. only library databases will be used).

Troubleshooting

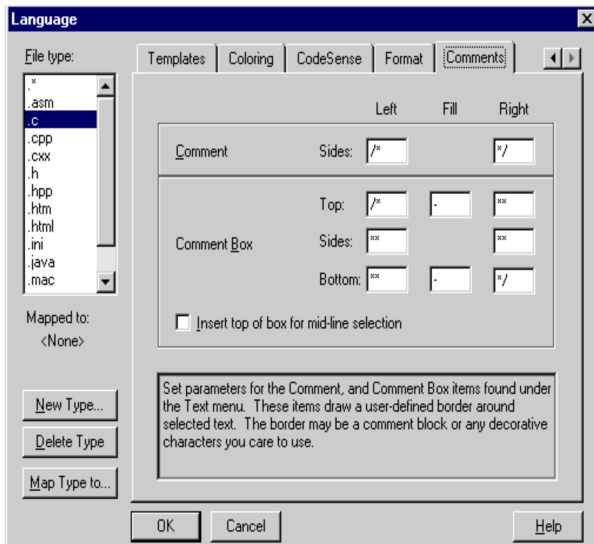
If CodeSense is not performing as expected, do the following:

- Make sure that the necessary CodeSense library databases are checked and have been properly scanned in **Customize | CodeSense Global Configuration**.
- Make sure that **CodeSense DLL** is marked for the appropriate file type (C/C++ or Java) in **Customize | Language | CodeSense** and that desired configuration options are marked.
- Use the `TURN_CODESENSE_OFF/ON` macros to prevent troublesome blocks of code from being scanned.
- If project files aren't being scanned, make sure that **Auto-update Symbols Database** is marked in **Project | Properties | Directories**.
- If some macro definitions are being parsed incorrectly, add them to the `[CodeSense Definitions]` section of the CodeSense configuration file `CWSENSE.INI`. See the online topic *CWSENSE.INI* for more information.
- If the correct symbol is not included in those scanned from CodeSense library databases, make sure that the file with the symbol is either included in the current CodeWright project, or included in the Symbols Window's file list.

Comments and Comment Boxes

There are two items, **Comment** and **Comment Box**, on the **Text** menu, which are used to automatically insert comments into your code. The **Comment** and **Comment Box** 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, and 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.

Customize|Language|Comments



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

With the ever-increasing popularity of the Internet, it has become necessary to own at least one good HTML editor. CodeWright provides several features that make it ideal for HTML editing. Those features are described next.

HTML Language Support

When HTML files are opened in CodeWright, ChromaCoding is immediately available. All HTML keywords, comments, and strings are colored instantly. Template expansion (see the topic *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 code-statements.

HTML Popup Menu

CodeWright's default right click-popup menu has a handy **<tag> attributes** option that displays applicable attributes for the closest open HTML tag. Just position your cursor close to an opening HTML tag, then press your right mouse button. The first option on the resulting popup menu will access a submenu containing a list of applicable tag attributes, if any.

Items on the popup menu can be changed by using the **Edit this menu** item, or by editing the HTML.MNU file directly (See the chapter *Custom Interface*).

WYSIWYG Editor/Viewer

CodeWright has an HTML WYSIWYG (What You See Is What You Get) editor/viewer that must be turned on in **Customize | Libraries** to be used. See the topic *Viewing and Editing Internet Files: Installation Instructions* for instructions about turning on WYSIWYG editing. See the topic *Using HTML WYSIWYG* for information about how the feature is used. CodeWright's HTML\WYSIWYG requires either Windows 95 osr2, Windows 98, Windows 2000, Windows Millennium, or Windows NT and Internet Explorer 5 or later.

HTML Viewing - Web Browser Interface

In addition (or alternative) to the WYSIWYG editor, CodeWright offers a **Web Browser as Viewer** interface that displays files being edited in certain installed web browser(s). See CodeWright's online help topic *HTML Web Browser Interface* for more information.

This viewing method must also be turned on in **Customize | Libraries**. The process for enabling both of CodeWright's HTML viewing and editing features is described next.

Viewing and Editing Internet Files: Installation Instructions





To install CodeWright's WYSIWYG HTML editor or the web browser interface, do the following:

- Go to **Customize|Libraries** and mark one of the following boxes:
 - ✓ **Web Browser as Viewer-** installs CWWEB.DLL, to enable CodeWright's web browser interface. See the online help topic *Web Browser Interface*.
 - ✓ **HTML WYSIWYG Editor/Viewer-** installs CWHTML.DLL, to enable CodeWright's WYSIWYG HTML editor.
- If none 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 CodeWright directory. Both installation methods cause CodeWright to load the library into memory. They also add lines to the CWRIGHT.INI file so that the DLL(s) will automatically load when CodeWright starts up. Examples of the entries in CWRIGHT.INI follow:

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


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.


HTML WYSIWYG editing provides the following capabilities:

- Edit HTML files as normal documents.
- Quickly insert HTML tables and graphics.
- Resize HTML tables and graphics with mouse drag operations.

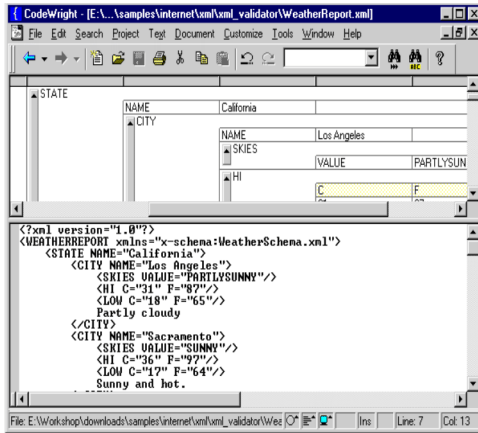
Note: By default, changes made in the code window must be saved or undone before editing in the WYSIWYG window. Conversely, changes made in the WYSIWYG window must be updated or abandoned before the code window can be edited. The **Update** button  transfers changes one-way, from the current window to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

If you wish to enable differencing/merging (so the **Update** button combines changes from both views and updates both views), press  to invoke the **HTML Configuration Options** dialog and mark **Enable Differencing/Merging**. Also on this dialog, mark **Open Viewer in Edit Mode** to open the WYSIWYG viewer in editing mode by default, or uncheck **Enable WYSIWYG Editing** to disable editing in the WYSIWYG window. You can also set these flags programatically using the **HTMLConfigFlags** API function, described under **Help | CodeWright API Library**.

XML Split Window Viewer

An **XML Split Window Viewer** option is available in CodeWright's **Customize | Libraries** dialog. Check it to enable CodeWright's XML viewer. The XML viewer takes up the top portion of a horizontally split edit window that contains a valid XML document. The bottom portion of the window displays the document's XML code. If the window is not already split it must be split manually. Windows can be split using vertical scrollbars, or by using the CUA key sequence .

XML Split Window Viewer



CodeWright's XML split window viewer displays XML code in a collapsible grid. The grid shows a hierarchy of the elements and element attributes in an XML document as they relate to the document's root element. The individual elements can be collapsed within the grid, so that only desired portions of the grid display. The grid can also be collapsed completely so that only the root element displays. Double click on any valid XML element or element attribute within the grid to position the text cursor at the corresponding element within the XML code. The code will then be ready for editing.

COBOL Editing

This section describes CodeWright's COBOL language support and editing features.

COBOL Lexer and DLL

CodeWright's COBOL ChromaCoding support is available via a DLL or a ChromaCoding lexer.

To use the DLL:

1. mark the option **COBOL** in **Customize|Libraries**.
2. Go to **Customize|Language**.
3. Select '**.cob**' in the **File type** list. If you do not see the correct file type, click **New Type**, and add it.
4. Select the **Coloring** tab.
5. Mark **DLL**.
6. Click **OK**.

To use the lexer:

1. Go to **Customize|Language|Coloring**.
2. Select '.cob' in the **File type** list. If you do not see the correct file type, click **New Type**, and add it.
3. Mark **Lexer**.
4. Choose **COBOL** from the drop down list.
5. Click **OK**.




Open a COBOL file to see ChromaCoding language support. Note that language support DLLs and ChromaCoding lexers can be used together. If both exist, it is advisable to use the lexer for ChromaCoding, and the DLL for other language support features.

COBOL Extensions

CodeWright has a **COBOL Extensions** module that provides various COBOL-specific functions and controls. To turn it on, mark **COBOL Extensions** in **Customize|Libraries**. The features that are provided by the module are listed next.




Resequence Line Numbers

Cobol Extensions provides a    keystroke to resequence line numbers. This works as follows:







1. Press   . A dialog will appear entitled **Enter Sequence Info**.
2. Enter the **Start** number for the line number sequence.
3. Enter the **Increment** number that the line number sequence will be incremented by.
4. Press **OK**. Line numbers will be resequenced according to the dialog's inputs. Note the following:
 - ✓ The **Start** number will be placed at the top of the file.
 - ✓ If numbers did not exist previously in the file, they will be inserted at the far-left margin of the file, taking up the first 6 columns of the line.
 - ✓ Line numbers will not be inserted if the file is not saved.
 - ✓ Line numbers will not be inserted if there is not more than one line in the file.

- ✓ If text already occurs at the position where a line number would be, the number will not be inserted. However, the number will still be counted.
- ✓ For non-numbered lines that have numbered lines above and below them, numbers will only insert on the lines between those lines. For non-numbered lines that have numbered lines below them, numbers will only insert up to those lines.

Line Number Handling when Lines are Copied or Moved

When **Cobol Extensions** is loaded, COBOL line numbers will become spaces at the locations at which they are copied or moved. This will keep the numbers from being repeated every time text is copied from one part of a file to another. New line numbers can then be inserted using the 'Resequence' key    (see above).

Toggle COBOL Comment

Cobol Extensions provides a   keystroke to toggle a COBOL comment delimiter. Press   to have a comment delimiter (*) inserted at column 7 of the current line. If a comment delimiter already exists in column 7, Press   to remove it.

Automatic Time/Date Stamp on Modified Lines







Time/date stamps are automatically inserted on modified lines at **File|Save**. Note the following:

- ✓ The line-modified (time/date) string is inserted at column 73 of the modified line.
- ✓ The <End> key will move the cursor to the end of the line of code, not to the end of the line-modified (time/date) string.
- ✓ Entering a new line will not move the line-modified string off of its designated line.
- ✓ The string that is inserted can be customized by modifying the date macros in the **History Template** section of the **CobolExt Settings** dialog (**Customize|CobolExt Settings**).

String Literals Automatically Continue on New Lines

COBOL Extensions automatically inserts "-" (dash-double-quote) on newlines when the previous line contains an open or continued string literal (no closing quote).

Validate Line Number Sequence




A    key sequence is provided to validate line number sequence. Press    to have CodeWright highlight the first line it finds that has an invalid line number, if the line exists. If it does exist, the following message will display in the status bar:

Sequence numbers are invalid.

If line numbers are valid, no lines will be highlighted, and the following will display in CodeWright's status bar:

Sequence numbers are valid.







Patch File Shows Changes in a File




COBOL Extensions provides a    key that creates a 'patch' file. The patch file shows which lines are different between the file that is being edited, and another version of the file. CodeWright locates the other version using path and filename information from the file being edited. The method CodeWright uses to extract path and filename information can be customized by modifying the filename component macros in **Customize | CobolExt Settings**.

For example, to create a **COBOL Extensions** patch file that will show the differences between the file being edited and the backup file that CodeWright optionally creates when the file is saved, set the following filename component macros in **Customize | CobolExt Settings**:

Patch File: %v%pPATCH\%r%e

Original Source File: %v%p\%r.bak

In a modified COBOL document press   . The above settings will create a patch file with the same name as the file being edited. The file will be located in the PATCH subdirectory of the file being edited. The patch file will not be created if the PATCH directory does not exist. The patch file contains a list of line numbers for any lines that are different between the edit file and the **Original Source File**. New text in the file being edited will also be recorded in the patch file along with the line numbers. If some of the lines in the file being edited do not have line numbers, a message will appear, and the patch file will not be created. Press    to resequence the line numbers.





The    key sequence will save the current file while creating the patch file. The contents of the patch file will be over-written every time the key sequence is successfully completed.



For a description of filename component macros, see the topic *Filename Component Macros* in the chapter *Projects, Project Spaces, and Workspaces*. For more information about the **CobolExt Settings** dialog, see CodeWright's online help topic *CobolExt Settings*.

Cobol Extensions does not require that the COBOL language support be turned on. However, the module will be more useful if it is.

Hex Editing

A frequently used CodeWright 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. To enter Hex mode:

- Press    in the CUA or the BRIEF keymap,
- Press  in the vi keymap.
- Use the button on the **Edit** toolbar (described briefly in the chapter on *Custom Interface*) that switches the current document to Hex mode.

When you enter the Hex mode, the  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  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 overtype mode. CodeWright can insert characters while in hex mode, and it will be in insert mode if you were in insert mode before entering Hex mode.

Note: 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. So, as you type, you will alternately see two nibbles inserted, and one nibble overtyped.

Handy Hex-Editing Tips and Features

Keep in mind the following CodeWright 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 the Chapter on *Search and Replace and Navigational Tools*.

Clipboard and Scrap Buffers

In CodeWright, copy/paste operations can be stored in the Windows Clipboard, or in CodeWright's Scrap Buffers. To toggle between Clipboards and Scrap Buffers, select one or the other from the **Edit** menu. In both cases, multiple buffers are allowed.

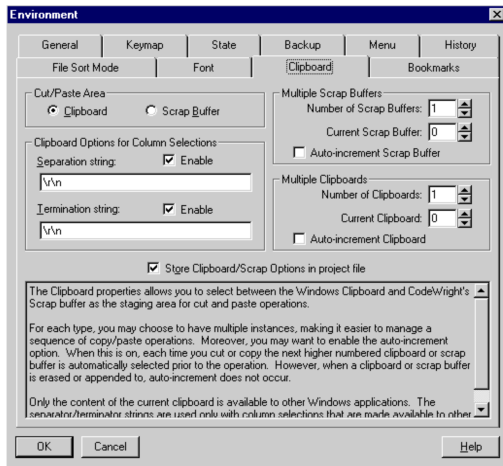
Scrap Buffers

CodeWright 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 CodeWright much as the Windows Clipboard is used for cut and paste operations between applications.

Multiple Clipboard/Scrap Buffers

CodeWright allows the use of multiple Clipboards or Scrap Buffers. The number of buffers available can be defined on the **Clipboard** tab of the **Customize|Environment** dialog. The default is one for each.

Customize|Environment|Clipboard



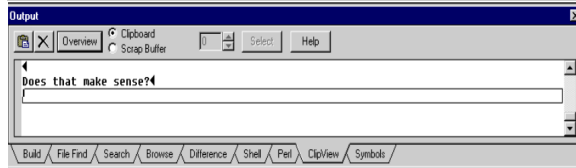
The **Clipboard** tab also has an **Auto-increment Clipboard/Scrap Buffer** option. With either enabled, the clipboard/scrap buffer to use is incremented immediately prior to copying or cutting new content. If the highest numbered clipboard/scrap buffer is already the current one, the zeroth one is used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

Check the option **Store Clipboard/Scrap Options in project file** 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.

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**. Make the **ClipView** tab of CodeWright's Output Window visible by marking the **Clipboard/Scrap Viewer** option in the **Customize|Libraries** dialog.

ClipView Tab



The **ClipView** tab provides the following functionality:

- **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.
- **Paste** Button: Pastes the contents of the ClipView window.
- **Delete** Button: Deletes the contents of the ClipView window.
- **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.
- **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.
- **Increment Clipboard/Buffer**: Change the number in this field to view the contents of another Clipboard or Scrap Buffer.

Additional information on CodeWright's ClipView tab is available in the online help.

API Assistant

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

Using the API Assistant

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, CodeWright will attempt to locate assistance for that word. If no word is nearby, a prompt comes up for a subroutine or function name to look up. If more than one occurrence of the word is found, the prompt offers a selection to choose from.

A form to be filled out is then presented. There is an edit box or set of checkboxes for each parameter used by the subroutine or function. The parameter type is listed next to each edit box or set of buttons. 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

The use of checkboxes in the API Assistant is 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.

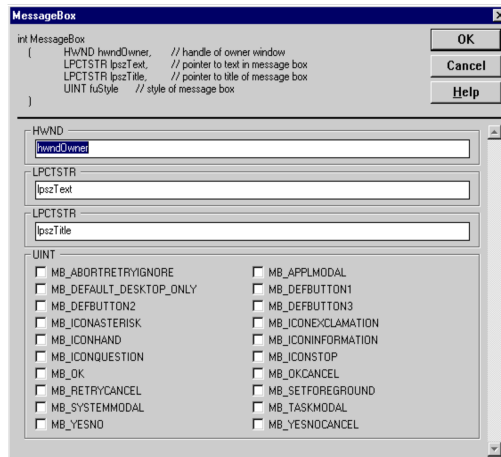
API Assistant Example

Refer to the following example on using the API Assistant:

The API Assistant could be used to help complete the **MessageBox** function, from the Windows32 API, in the following way:

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

The API Assistant (MessageBox) Dialog



3. Fill in the necessary parameters and mark the flags to be inserted.
4. Once the desired parameters have been inserted and the necessary flags have been chosen, click **OK** to insert the **MessageBox** function into the current file.

Currently, API Assistance is provided for the Windows API, Standard C Library functions, Microsoft Foundation Classes, Delphi's API, Java and Sun Java, HTML, and the CodeWright API.

API Assistant Databases

The API Assistant gets its information from databases. The databases supplied as of this writing are described in the table below:

Filename	Description
WINAPI.TDB	Contains information about Windows API function calls.
CWAPI.TDB	Contains information about CodeWright API function calls.
MFC.TDB	Contains information about Microsoft Foundation Classes.
C.TDB	Contains information about Standard C library functions.
JAVA.TDB	Contains information about Java functions.
SUNJAVA.TDB	Contains information about Sun Java functions.

Filename	Description
HTML.TDB	Contains information HTML tags.
DELPHI.TDB	Contains information about Delphi functions.

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. More serious modifications may best be done with the automation tools described below.


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.

Automation Tools

A tool has been provided to take data, in one of several formats, and make entries in an API Assistant database. This enables automatic creation and update of API Assistant databases for APIs or libraries for which support is not provided.

A method needs to be created for getting the data into the proper format. This might be done using an AWK script on source files or DDE queries to InfoView. After getting the data in the proper format, the automation tool will do the work of creating or updating the database. A description of the automation tools is contained in the CodeWright's online help, under the topic *API Assistant*.

Using Help in CodeWright

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


Indexing and Accessing Help Files

CodeWright ships with its own help files (CWHELP.HLP, CWAPI.HLP, and CWAPPBAS.HLP). You can also index or access help files from other software packages, so that they can be used from within CodeWright.

Help files that can be accessed from within CodeWright must be one of the following:


- .HLP Windows help files
- Microsoft Visual C++ 6.0 HTML help files
- .IVT Microsoft Visual C++ 5.0 help files
- .MVB Microsoft C++ 4.0 help files
- .CHM Compiled Microsoft HTML Help Files

Indexing .HLP Help Files

To index your .HLP help files, select **Add** in the **Help | Configure Index** dialog, and then browse for the help file you need. When you have selected it, choose **Close**. CodeWright will add this help file to the help index (CWRIGHT.IDX). CodeWright also has a **Help Index File Wizard** that will combine selected .IDX files with CodeWright's .IDX file, thereby creating a more comprehensive help index with minimal effort. The wizard can be accessed from the **Configuration Wizard Choices** dialog on the **Help** menu. Once the index has been created, press  on a word that is contained in the newly-indexed help file to have the word 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 then be selected and a second search can be initiated.


CodeWright can also be set up to display help from files other than .HLP files (i.e. .MVB, .IVT, .CHM and MSVC 6.0 .HTML help files). These processes will be discussed next.

Accessing 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. CodeWright provides the ability to access MSVC's context sensitive HTML help using the  key.

To set CodeWright up to access HTML help files, go to the **Configure Help Index** dialog:

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.

Accessing Compiled Microsoft HTML (.CHM) Help Files

Compiled Microsoft HTML files can be viewed with an HTML Help Viewer that comes with Visual Studio. Windows must have a file association for .CHM files to the executable `hh.exe` for CodeWright's help system to display a .CHM file.

To set CodeWright up to access .CHM help files, go to the **Configure Help Index** dialog:

1. Press **Add**. The **Add Helpfile Keywords** dialog displays.
2. Browse for the desired .CHM file. You will probably need to select **All Files** in the **Files of Type** drop-down list to see .CHM files.
3. Select the .CHM file and press **Open**. The file will now display in the **Filename** window on the **Configure Help Index** dialog.
4. With the file still highlighted, press **Edit**.
5. On the **Edit File Information** dialog, type a **Description** to identify the .CHM help file in your list.
6. Press **OK** to exit back to the **Configure Help Index** dialog, and **Save** to retain your changes.

To access information from the indexed .CHM file, select **Help | Search for Help On**. Type the **Keyword** to search for, select the desired entry from the .CHM file, and press **OK**.

Accessing MSVC 5.0 (.IVT) Help Files

MSVC 5.0's Help format uses .IVT files known as InfoViewer Titles (formerly .MVB files). These are read by the online MSVC system known as InfoViewer. InfoViewer is built in to MSDEV.EXE, and is available stand-alone on the MSDN Library-Visual Studio 97 CD, as IV5.EXE. If you have not installed the entire MSDN Library on your drive, then you will need the MSDN CD in your CD drive when accessing help.

Complete the following steps in CodeWright:

1. In the **Customize | Library** dialog, under **CodeWright Libraries**, you will find **InfoViewer Titles**. Check the box, and click **OK**.
2. Go to **Help | Configure Index File**.


3. Click the **IVT Setup** button, which allows you to setup InfoViewer as your default help engine. If the button is disabled, then you need to de-select your MVB viewer (click **MVB Setup**, uncheck **Use as default when keyword not found**, and click **OK**). CodeWright can't use both .MVB and .IVT help files concurrently.
4. On the **InfoViewer Title Setup** dialog, enter the name of the program used to view the .IVT files (msdev.exe, or iv5.exe).
5. Check **Use as default when keyword not found**.
6. Select the search method you would like to employ (index, or full-text search).
7. Click **OK**, and finally click **Close**.

Now when you bring up the **Find Help** dialog via **Help | Search for Help On**, you'll see **Use Installed InfoViewer Titles** in the dialog listbox. Click **OK**, and CodeWright will switch to InfoViewer and bring up the Search dialog with the appropriate keyword. InfoViewer needs to be running to receive messages from CodeWright. CodeWright will launch the InfoViewer program if it is not running.

Accessing MSVC 4.0 (.MVB) Help Files

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

1. Go to **Help | Configure Index File** and click **MBV Setup**.
2. In this dialog:
 - Uncheck the **Use Default Help** box.
 - Uncheck the **Use DDE** box.
 - Remove the filename for the default .MVB file from the **Topic** field.
 - Enter the following macro in the **Topic** field: `$(MVBFILE)`.
3. Return to the **Help | Configure Index File** dialog.
4. Press the **Add** button. You will see that CodeWright now allows you to index multiple .HLP and .MVB files. Add any necessary .MVB files.

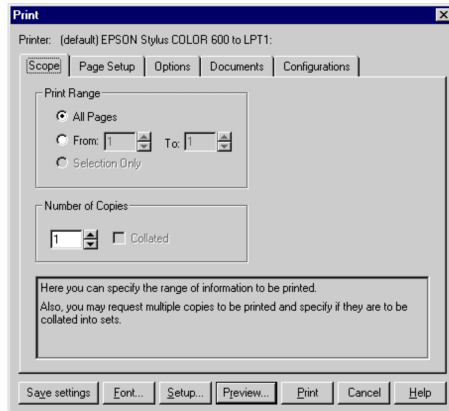
Once the .MVB help integration setup is complete, position the cursor on a keyword and press  or enter the keyword in the **Search for Help On** dialog, to access all the help (.HLP and .MVB) files listed in the index. A dialog box showing the help files found will be presented from which to choose the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

The next section talks about some of CodeWright's more unique print features.

Printing

CodeWright'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 CodeWright's print dialog is available below.

Print Dialog



Print Configurations

CodeWright 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, and 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 CodeWright'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 CodeWright 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|Colors** dialog.

Print Preview

CodeWright 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

CodeWright'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 checkbox 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.

Print Headers and Footers

The header and footer groups on the **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.

Example:

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

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

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

```
C:\CW32\FOO.C
```

```
1\1\1999
```

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

```
;%P
```

Macros to be used in headers and footers are listed in the following table:

Macro	Expansion Value
%d or %D	date
%t or %T	time
%f or %F	filename
%v or %V	volume
%r or %R	root

Macro	Expansion Value
%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

7- Projects, Project Spaces, and Workspaces

This chapter describes how to use the CodeWright project, project space, and workspace facility. The complex relationships between files and groups of files that are routine in program development demand the type of organization that can be achieved with CodeWright projects.

Definitions

Projects, project spaces, and workspaces are defined as follows:

What is a Project?

A CodeWright project is a single file containing a list of filenames and settings that are intended to make up the project. The contents of a CodeWright project are at the discretion of the person making it. The configuration file containing the project files and settings is stored in the directory in which the project was created, and 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 CodeWright 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, CodeWright 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.

What is a Project Space?

Project spaces store and display sets of projects. Before a project can be made, a project space has to be set up. 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 has the **Add New**, **Add Existing**, and **Remove Project(s)** menu items for adding new or existing, or for 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 and their version control status.

What is a Workspace?

A workspace can be thought of as a separate instance of CodeWright. It is a "mini-project" within a project. When you change workspaces, you have the ability to pick up where you left off with a group of files. It doesn't matter if you worked on that group of files a half-hour before or three months ago. It is as if an instance of CodeWright were frozen in time containing these files.

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 CodeWright state file that is swapped on the fly.

- ✓ For additional discussion of workspaces, refer to the topic *Creating, Selecting and Saving Workspaces*, in this chapter.

Creating a Project Space

Before projects can be created, it is necessary to create a project space that will contain the projects. Multiple project spaces can also be created, but only one can be open at a time. To create a project space, do the following:

1. Click **Project|Project Space|New**.
 2. Click **Browse** to make sure that the project space is being created in an appropriate directory.
 3. Type in a name for the project space in the **Filename** box. Keep in mind the following:
 - The **Close existing documents/windows** option causes all open windows to be closed when the project is created.
 - 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 (CodeWright project space) file type extension is automatically appended to the inserted name.

- ✓ The contents of the corresponding .DSW file will be read to create 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 CodeWright 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.
- CodeWright assumes the filename extension .PSP for project space configuration files if one is not supplied. You may specify another extension, if desired.
4. 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 the following dialog.

Project | Properties | Members

Projects are added to project spaces, and then files to projects, in the **Project|Properties|Members** dialog. The information for creating and adding files to projects is contained in the section on *Creating a Project and the Members tab of Project Properties*. Before covering that topic, however, it is necessary to have some understanding of the Project Properties List and the process for setting project configurations. Those topics will be covered next.

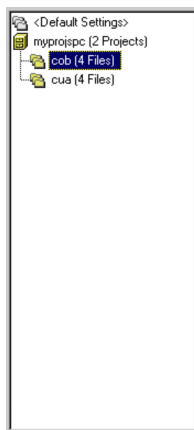
The Project Properties List and Project Settings

CodeWright's **Project|Properties** dialog is where many necessary CodeWright configuration settings are made. One item in the dialog, the Project Properties List, is important to all tabs of the dialog. The next few paragraphs talk about the Project Properties List, and how the items in the box affect the rest of the dialog.

Project Properties List

The Project Properties List is available from all tabs of the **Project|Properties** 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|Properties** dialog changes depending on the item selected.

Project Properties List



Default Settings

The **<Default Settings>** item in the **Project Properties List** of the **Project Properties** dialog is available at all times, regardless of whether or not there is a project or project space open. 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|Properties** dialog are only available when the **<Default Settings>** item is selected; others are only available when projects or project spaces are selected. The **Members** tab of the **Project|Properties** dialog, for example, is used for adding projects to spaces and/or files to projects. It is therefore not available when the **<Default Settings>** item is selected, since the **<Default Settings>** item does not store projects or files. The **Members** tab is the only tab that is not used for configuration settings, however. It is therefore the only tab that is available at the project-space level, since 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 because they are used for making configuration changes that can be either global (i.e. **<Default Settings>**) or specific to individual projects.

Working Directory

One important item to consider when setting project defaults is the project's working directory. It is set in the **Project|Properties|Directories** dialog. The working directory is the directory from which all tool commands (configured in the **Project|Properties|Tools** dialog) are run.

The default-working directory is set to `%x`, a filename component macro that expands to the directory containing the project's configuration file. You can select **Working Directory** in the Directories list and press **Browse** to change the working directory on the **Project Working Directory** dialog. (More information on the **Tools** tab and filename component macros can be found under the topic *Tools Tab of Project Properties*, 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|Properties** dialog, will be covered throughout the remainder of this chapter.

Creating a Project and the Members Tab of Project Properties

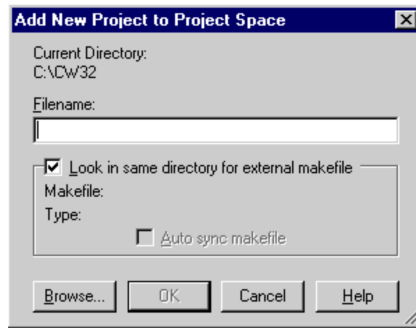
Once a project space has been created, you can add new or existing projects to it. This can be done from the **Project|Project Space** menu, or the **Project|Properties|Members** dialog. **Project|Properties|Members** should appear immediately after a new project space has been created. Otherwise the dialog can be accessed via the **Project** menu.

Creating a New Project

To create a CodeWright project in **Project|Properties|Members**, complete the following steps:

1. Highlight the current project space in the Project Properties List on the left.
2. Click  to access the **Add New Project to Project Space** dialog. (The same dialog can be accessed using **Add New Project** on the **Project|Project Space** submenu.) In the **Add New Project to Project Space** dialog, click **Browse** to make sure that the directory the project is being created in is appropriate.

Add New Project





3. Give the project a name. Keep in mind the following:
 - The **Look in same directory for external makefile** option will automatically insert the name of any similarly titled makefile that resides in the same directory. Turn off the option if you don't want makefile names inserted in the **Filename** box. For more information on what this option does, read the topic *Reading External Makefiles and Workspaces*.
 - The **Auto sync makefile** option tells CodeWright to synchronize any selected external makefile with the project being created, so that files that are added to the makefile are automatically added to the CodeWright project.
 - CodeWright automatically uses a .PJT extension for project files.
4. Click **OK**. CodeWright creates a new project configuration file in the chosen directory. The new project is listed in the Project Properties List, under the current project space.

Adding Files to a Project




Files can be added once the project has been created. Files are added in the **Project|Properties|Members** dialog. To add files do the following:

1. Highlight a project to which the files will be added in the Project Properties List.
2. Add the files using one of the following methods:

- Click  to add a new file to the project.
- Click . Add individual files by typing in their names, or add multiple files using file filters (e.g. *.C). Use the [...] button to browse for different directories.

Note: 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. (e.g. *.C;*.H;*.CPP;C:\TEST*.C;C:\TEST*.H).

The **Include Subdirectories** checkbox causes matching files in subdirectories to also be included.

- Use  to browse for files. Add individual files by double clicking or by typing in the names. Add multiple files by  or -clicking.
- Use the **External Makefile** option to read files from an existing makefile into the project being created (see the topic on *Reading External Makefiles and Workspaces*).
- Use the **VCS Project** option to 'read' a version-control project file (see *Using Version Control in CodeWright*, in the chapter *Version Control*).

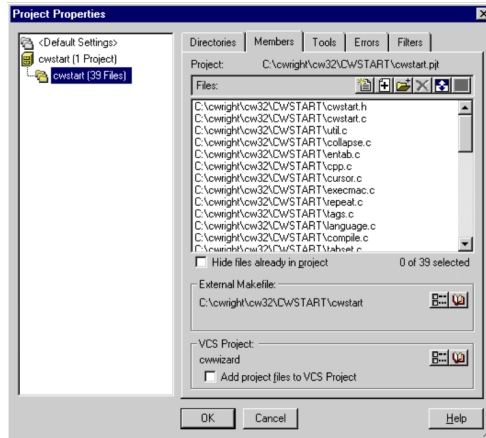
3. Click **OK**. Any matched files should be immediately added to the project.

Files that are included in the project are listed in the **Files** list box in the center of the dialog. To delete files from the project, select them in the box and press the **Delete** button.



Adding Existing Projects to a Space

Thus far we have discussed adding new projects to a project space, but we have not covered the process for adding projects that already exist. That process will be described here. To add existing projects to the current project space, access the **Project|Properties|Members** dialog.

Project Properties Members



Complete the following steps:

1. Highlight the current project space in the Project Properties List.
2. Add projects using one of the following methods:
 - Click  to access the **Select One or More Projects to Add to Project Space** dialog (also accessed using the **Add Project...** item on the **Project|Project Space** menu). Browse for individual project files or use extended selection listbox rules (i.e. SHIFT/CTRL keys with mouse clicks) to add multiple projects from the same directory at a time.
 - From the **Read External Workspace** dialog (accessed in the **Project|Properties|Members** dialog) select an external Visual Studio workspace that CodeWright will use to **Read**  projects from. The projects will be read from the Visual Studio workspace into the current CodeWright project space. See the topic *Reading External Makefiles and Workspaces* for more information.
3. Click **OK**. Appropriate projects should be immediately added to the project space. The projects are shown below the current project space in the **Project Properties List**.

Auto Detect File Type to Load or Create Projects and Project Spaces

The option **Auto-detect file type** in CodeWright's **Customize | Environment | General** dialog allows CodeWright projects and project spaces to be opened in any way that any file can be opened in CodeWright (e.g. **File | Open**, drag and drop, opening from Project Window, opening from FTP dialog, etc). When the option is marked, the following will occur when opening designated files with any allowable file-open method:

- Files with .PSP file type extensions (CodeWright project space) will load as CodeWright project spaces.
- Files with .DSW file type extensions (Visual Studio workspace) will create CodeWright project spaces with autosync enabled (see *Synchronize Makefile/Workspace with Project/Project Space*).
- Files with .PJT file type extensions (CodeWright project) will load as CodeWright projects.
- Files with .DSP file type extensions (Visual Studio project) will create CodeWright projects with autosync enabled (see *Synchronize Makefile/Workspace with Project/Project Space*).
 - ✓ Any supported makefile will create a CodeWright project. See the drop down list under **Makefile Type** in the **External Makefile** dialog for a list of supported makefiles.
 - ✓ Do not mark **Auto-detect file type** if you want CodeWright to load these files for editing only.
- CodeWright will always override **Auto-detect file type** if a project file is opened from the Output Window's **Build** or **Search** tabs. Project files will always be opened as text files from these windows.


If the option **Auto-detect file type** is not marked, the files listed above will open as normal edit buffers in CodeWright.


Reading External Makefiles and Visual Studio Workspaces

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

To have CodeWright automatically use this capability, mark the **Look in directory for external makefile/workspace** option in the **Create a New Project Space** dialog, or in the **Add New Project to Project Space** dialog. CodeWright 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, CodeWright automatically inserts the name of the file into the **Create a New Project Space** dialog, or the **Add New Project to Project Space** dialog. The projects or files that are part of the workspace or makefile are subsequently read into the CodeWright project or project space.



To have CodeWright manually read files or projects from selected makefiles or workspaces into CodeWright projects or project spaces, do the following:

1. To read a workspace, select the current project space in CodeWright's Project Properties List. To read a makefile, select a project from the same list box.
2. Select  to bring up either the **Read User Makefile** or **Read External Workspace** dialog.

Note: By default, CodeWright knows how to read Visual Studio 5.0 and 6.0 workspaces and project files (.DSW and .DSP), and Microsoft Visual C++ , Borland, and CodeWright makefiles. These are listed in the drop down list box under the **Makefile** or **Workspace Type** combos.
3. Select the type of parser from the **Makefile/Workspace type** combo drop-downs to use for parsing the filenames from the respective file. Keep in mind the following:
 - 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*.)
4. Press **OK**.
5. In the **Project|Properties|Members** dialog, press . The project or project space will then 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. In the Project Properties List, highlight a project or a project space.
2. Click  in the **External Workspace/Makefile** section.
3. Browse for the appropriate makefile or workspace file using the **Browse** button next to the **Workspace/Makefile Name** edit box in the **Read External Workspace/Makefile** dialog.
4. In the **Makefile/Workspace Type** edit box, give the parser a descriptive name.
5. Enter an applicable regular expression for the **Regular Expression Parser**.
6. If you want to synchronize the makefile or **workspace** with the CodeWright project or project space (i.e. update the CodeWright project/project space with a new file/project when a new file or project is added to the makefile or workspace), mark **Auto Sync Workspace/Makefile** (described next).
7. Click **OK**.
8. Click **Read** .
9. Click **OK**. Files or projects in the makefile or workspace are added to the project or project space.

Creating a new makefile or workspace parser requires some knowledge of regular expressions. See *Regular Expressions* in the chapter *Search and Replace and Navigational Tools*.

Synchronize Makefile/Workspace with Project/Project Space

CodeWright's makefile/ workspace reader provides an auto-synchronize feature that will synchronize the projects and project spaces with associated external makefiles and workspaces. The option is called **Auto Sync**, and is available in the **Create a New Project Space**, **Add New Project to Project Space**, **Read External Workspace** and **Read User Makefile** dialogs. When enabled, Auto Sync Makefile/Workspace will automatically add new files and projects to CodeWright projects and project spaces when corresponding makefiles or workspaces have been updated with those files. The makefile or workspace can be any of the predefined types in the **Makefile/Workspace Type** drop down lists.

When **Auto Sync** is enabled for either makefiles or **workspaces**, all the items in the

Project|Properties|Members dialog are disabled except for . The ability to manually add files or projects is not allowed because the files are taken from the makefile or workspace. Any manually added or deleted files or projects are eliminated from the project when the sync occurs.

Note: Version Control Management items are not affected.

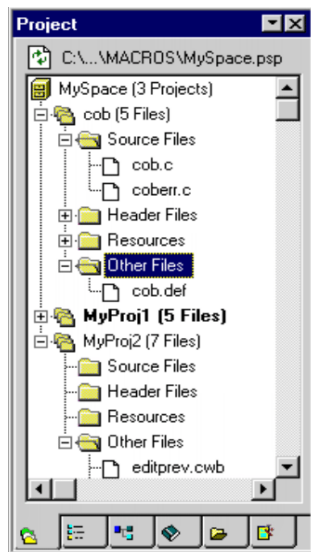
Characteristics of the File View Tab

The Project Window's **File View** tab displays projects and project spaces. By default, the Project Window is displayed on the left-hand side of the CodeWright 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 CodeWright. 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|Properties|Filters** dialog. Each filter is displayed as a folder containing files of the appropriate extensions. (Refer to the topic *The Filters Tab of Project Properties*, in this chapter, for more information on project filters).




To open a file for editing, double-click on the file in the **File View** tab.


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, CodeWright conveniently places a red checkmark 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, CodeWright will display the file with a gray checkmark in the project window.
-  Some Version Control projects are set to delete work files when they are checked in. CodeWright 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, CodeWright gives the corresponding file icon a slightly different color.

Note: Press the Refresh button  from time to time when using the Project Window to see the most current status of the files.

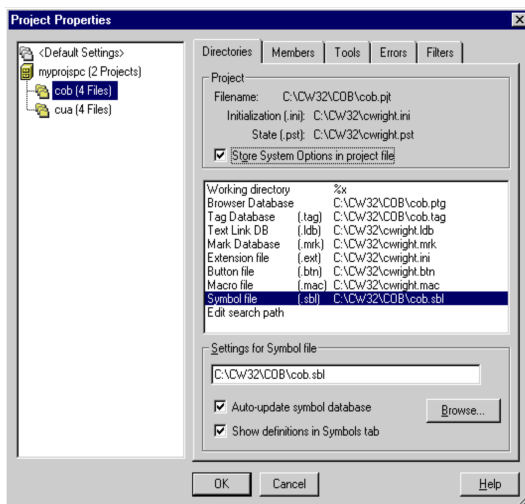
The primary purpose of the Project Window is to open files and facilitate version control operations. Right-click on a file, or on a selected group of files, in the Project Window to bring up a menu listing various operations that can be performed on the file(s). Additional right-click popup menus are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

Directories Tab of Project Properties

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

1. Access the **Project|Properties** dialog.
2. In the Project Properties List:
 - Select a project if the directories being viewed or modified are intended to be specific to the selected project only.
 - Select **<Default Settings>** if the directories being viewed or modified are intended to be global.
3. Click on the **Directories** tab. Most of the directories and filenames listed in this dialog can be made project-specific, but only if the **System Options** checkbox is marked. Once **System Options** 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 unique names for the selected project.

Project|Properties|Directories



A brief description of each of the directory and file listings in **Project|Properties|Directories** follows:

- The **Working Directory** is the directory from which commands set on the **Tools** tab of the **Project|Properties** 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 the chapter on *Search and Replace and Navigational Tools*.
- The **Tag Database** indicates the name and location of the file that contains tags information. More information about tags can be found in the chapter on *Search and Replace and Navigational Tools*.
- 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 the chapter on *Search and Replace and Navigational Tools*.
- The **Mark Database** indicates the name and location of the file that stores bookmark information. More information about bookmarks can be found in the chapter on *Search and Replace and Navigational Tools*.
- 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 CodeWright'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 the chapter on *Extend CodeWright*.
- 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 (i.e. <project_name>.SBL). If an existing project's symbol file is CWRIGHT.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 CodeWright'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 *Search and Replace and 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 CodeWright'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 CodeWright 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.

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

Storing Configuration Options with a Project

CodeWright 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|Properties** dialog while a project is selected. Others will only be stored with the project if certain options are marked in various CodeWright dialogs. The settings that may be stored with projects are listed next:

- System Options: Mark **System Options** for selected projects in **Project|Properties|Directories**.
- Auto-save options: Mark **Store Auto-save Options in the 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 |Properties|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| Properties|Tools** while the desired project is selected.

In addition to indicating that certain system options be stored with the project, the **System Options** checkbox in **Project|Properties|Directories** allows the listed configuration file names 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 **System Options** checkbox on the appropriate version of the **Directories** tab, complete the following steps:

1. Access the **Project|Properties** dialog.
2. In the Project Properties List:
 - 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 **Systems Options** checkbox, 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 CodeWright read configuration settings from a configuration file using an option on the **Customize** menu called **Read Configuration Data** The configuration information gathered will be placed in CodeWright's main configuration file or in a CodeWright project file, depending on the data being read and depending on CodeWright's current state (i.e. whether a project is open or not).

The series of checkboxes in the **Read Configuration Data** dialog limit what settings are read from the file. When marked, the **Update Configuration Files** checkbox immediately updates the project and/or configuration files with the specified information.

Tools Tab of Project Properties

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 CodeWright. These tools can be made project-specific or global.

To access/edit the appropriate version of the **Tools** tab, complete the following:

1. Access the **Project|Properties** dialog.
2. In the Project Properties List:
 - Select a project if the tools being viewed or modified are intended to be specific to the selected project only.
 - Choose **<Default Settings>** if the tools being viewed or modified are intended to be global.
3. Click on the **Tools** tab.

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. CodeWright doesn't come with the utilities (e.g. compilers and version control); these must be purchased separately. CodeWright 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 CodeWright that are used to run the utilities. When one of the menu items or buttons is clicked, CodeWright runs the tool by shelling to DOS and running the tool's command. Any output generated is captured for display in designated tabs of CodeWright's Output Window. The specific tool categories are:

- Build
- Compile
- Custom
- VCS

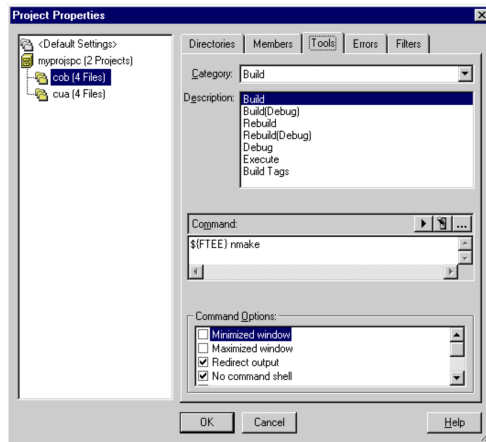
The following sections describe the tool categories in **Project**| **Properties**| **Tools**.

Build Tools

The **Build** category of tools in **Project**| **Properties**| **Tools** consists of the following:

- Build
- Build(Debug)
- Rebuild
- Rebuild(Debug)
- Debug
- Execute
- Build Tags

Project Properties Tools - Build Category



These tools (with the exception of Build Tags) should be set with any make and execute utilities necessary for building or running the files being edited in CodeWright. Filename component and symbolic macros can be used as placeholders that will insert or transform filenames and filename components (directories, filenames, etc) that need to be used with the utilities. This eliminates the need to manually insert filenames every time a new file is opened in CodeWright for editing or compiling. More information on filename component macros is provided in the section on *Filename Component Macros*.

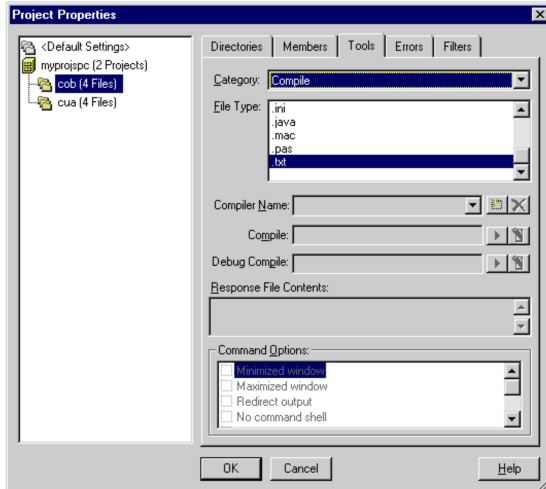
The **Build Tags** tool in the **Build** category does not need to be configured like the other tools. It comes pre-set with the appropriate command needed to run the Tags utility provided with CodeWright.

Compile Tools

The **Compile** category of tools in **Project|Properties|Tools** consists of the following:

- Compile
- Debug Compile

Project Properties Tools – Compile Category



The **Compile** tools should be set with any compile utilities necessary for compiling the files being edited in CodeWright. Filename component and symbolic macros can be used as placeholders that will insert or transform filenames and filename components (directories, filenames, etc) that need to be used with the utilities. This eliminates the need to manually insert filenames every time a new file is opened in CodeWright for editing or compiling. More information on filename component and symbolic macros can be found in the sections on *Filename Component Macros* and *Symbolic Macros*.

The **Compile** tools are the most commonly used tools in the **Project|Properties|Tools** dialog. Therefore, the instructions for setting them up are given their own chapter, called *Set up a Compiler*. Many of the concepts described therein apply to the other tool-categories as well. Refer to it for more detailed information on setting up **Compile** and other project tools.

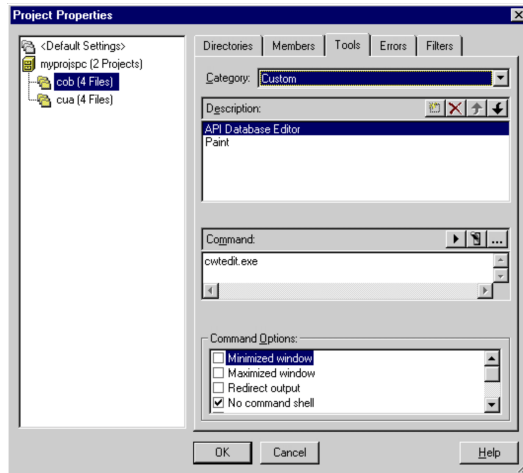
Response File Contents

The **Response File Contents** edit box in the **Compile** category of tools is used for commands that may be too long or that may be better used in a batch file. It only comes into play if the %Q macro is being used on one or both of the compile command lines. See the topic *Response File* for more information.

Custom Tools

The **Custom** category of tools is provided for running any DOS program from within CodeWright. Adding a custom tool in **Project|Properties|Tools** will add it to a list at the bottom of CodeWright's **Tools** menu. Clicking on a tool's name in the menu will then run the tool's associated program. Two custom tools are already set in CodeWright: Windows's Paintbrush program, and the API Database Editor, a tool used for editing CodeWright's API Assistant databases.

Project Properties Tools – Custom Category



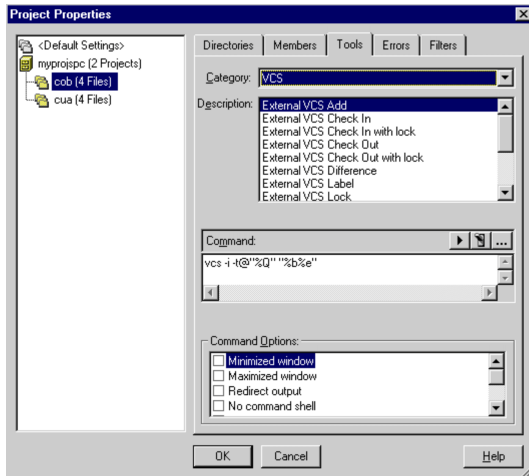
To make a new custom tool:

1. Select **Project|Properties|Tools**.
2. Choose **Custom** from the list of categories in the list under the **Categories** combo box.
3. Click on **Add**.
4. Give the custom tool a name.
5. Click **OK**.
6. Highlight the new custom tool in the listbox.
7. In the **Command:** edit box, type the appropriate command for running the intended program. (To make a tool that runs a CodeWright API, type the API in the **Command** box, and precede it with a !.)
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|Properties|Tools** dialog consists of a number of tools associated with various version control operations; these operations are available on different menus, buttons and dialogs in CodeWright. Set the tools with the commands that will run any command-line version control utilities used to perform version control operations (e.g. GETEXE or PUTEXE).

Project Properties Tools – 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 (directories, filenames, etc) used with the version control utilities. This eliminates the need to manually insert filenames every time a new file is opened in CodeWright for editing or compiling. 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 GETEXE 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 CodeWright. 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 *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 (i.e. **Build**, **Compile**, **Custom**, etc).
2. Highlight or choose the tool to be configured.
3. Enter the appropriate command in the **Command**: edit box at the bottom of the dialog. (To make a tool that runs a CodeWright API, type the API in the **Command** box, and precede it with a !.)

Note: The preceding steps are based on the assumption that the utility being launched will run 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 any utilities being used must be inserted with the command when performing step number 3.

For more detailed information about setting and configuring project tools, see the chapter *Set up a Compiler*. Although the chapter specifically talks about setting up tools in the **Compile** category, the concepts generally apply to all categories of project tools.

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 **Build** tool for the first time shows the following command in the **Command**: edit box:

```
${FTEE} nmake CFG="%x - Win32 Release"
```

Nmake is Microsoft's make utility, which may not be the utility of choice for the files being edited. If so, it will be necessary to change the command appropriately, to apply the obligatory utilities for the files being edited.

The **\${FTEE}** macro expands the full path and filename of FTEE.EXE, a utility that pipes output to CodeWright's output window. A detailed description of FTEE can be found in the chapter on *Set up a Compiler*.

Command Options on the Tools Tab

Each category of tools in the **Project|Properties|Tools** dialog has a number of options that affect what happens when any of the various tools are run.

Example: To control how CodeWright 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 the **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:

- **Redirect Output** appends the following items to the end of the command that runs the tool's 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. CodeWright then uses an FTEE program to display that information in the **Build** tab of the Output Window. (More information about FTEE can be found in the chapter on *Set up a Compiler*.)

- **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 CodeWright 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 CodeWright'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 filenames will vary as files are opened and closed. For this reason CodeWright provides a number of **Filename Component Macros** 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 CodeWright.

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 popup menu listing available macros. To access the popup menu, click the black right-arrow on the gray portion of the **Command** edit box.



A sample Macro Assistant popup menu is provided:

Macro Assistant

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

A list of these macros and the filename specs they represent are listed in the following table. 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 CodeWright 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.

Filename Component Macros	
Components	Description
%%	<i>Percent sign</i> Since component macros begin with a percent sign, you must use two percent signs to represent a literal % in your command.
%b	<i>Basename</i> The complete workfile name, less the extension. (e.g. C:\TEST\FOO)
%d	<i>Directory</i> The directory component of the workfile. This component has no filename and no drive component. It also has no trailing backslash. (e.g. \TEST)
%e	<i>Extension</i> The extension of the workfile. This component begins with a dot (.) unless the extension is null. (e.g. .C)
%n	<i>Project Name</i> The name of the version control project, if one has been defined and selected. If a project file is in use, this will expand to the path of the project file. Otherwise, this macro expands to a null string. (e.g. C:\TEST\VCSPROJ)
%p	<i>Path</i> The path component of the workfile. This component has no filename and no drive component. It has a trailing backslash. (e.g. \TEST)
%r	<i>Root</i> The filename of the workfile, less the extension. (e.g. FOO)
%v	<i>Volume</i> The drive component of the workfile, consisting of a letter and a colon. (e.g. C:)
%x	<i>Project Path</i> The path component of the project file currently in use (usually indicates the project directory). (e.g. C:\TEST)
%y	<i>Project Root</i> The base filename of the project file currently in use. (e.g. FOOPROJ)
%Q	<i>Response File</i> A temporary response file containing text that will be used with or for the command being configured. This macro is case sensitive. You must use an upper case Q. See the default check-in command for an example of its use.

Response File

The **%Q** macro and the **Response File Contents** edit box (on the **Project | Properties | Tools** dialog) work together to automate response files and avoid command line length limits. The **Response File Contents** edit box only comes into play if you have used the **%Q** macro on one or both of your compile command lines.

Note: This macro is case sensitive. An uppercase **Q** must be used.

When the **%Q** macro is used in a compile command, the name of a temporary file is substituted at that position in the command line. Before execution of the command, the contents of the **Response File Contents** edit box are written to that temporary file.

Example: `cl /c @%Q`

An alternate syntax for the **%Q** macro allows this macro to be the only thing on the command line. If the character following the **%Q** is a dot, the three characters following the dot are used as the temporary filename's extension.

Example: `%Q.BAT`

This syntax will cause the temporary file to be a batch file that can be executed alone. Most of the filename component macros can be used in other tabs of the **Project | Properties** dialog as well.

Example: **%x** is used to specify the project's working directory in **Project | Properties | Directories**.

%b is used to specify the base name of the Error File listed in **Project | Properties | Directories**.

Symbolic Macros

Symbolic macros are another form of macro that can be used with many of the commands in the **Project | Properties | Tools** dialog. CodeWright'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 CodeWright's Output Window (described in the chapter on *Set up a Compiler*). A list of Symbolic Macros and their descriptions appear next:

Symbolic Macros	
Macro	Description
\$ (BROWSEFILE)	Browser file as defined in the Directories tab of the Project Properties 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)	CodeWright's home directory.
\$ (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, Configure Index... item)
\$ (TAGFILE)	Tag file name as defined in Project Properties .
\$ (USER)	User ID defined in Tools Version Control Setup .
\$ (VCSLABEL)	Revision label as defined in Tools Version Control Maintenance .
\$ (WTAGS)	Starbase's command for generating a tags database. This command varies according to platform.

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

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

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 CodeWright's **Build** window (a tab on the Output Window).

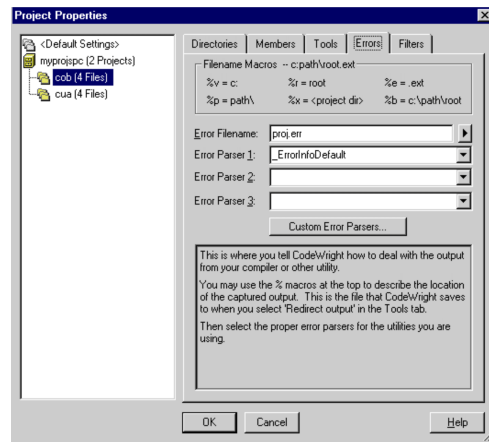
CodeWright uses two methods for capturing and displaying output from a compile. These methods, called FTEE and VDOS, are described in the next chapter on *Set up a Compiler*. Once the output has been captured, CodeWright uses Error Parsers to access the files that contain the errors in that output.

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. CodeWright 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|Properties** dialog.
2. In the Project Properties List:
 - Select a project if the error configurations being viewed or modified are intended to be specific to the selected project only, OR
 - Select **<Default Settings>** if the error configurations being viewed or modified are intended to be global.
3. Click on the **Errors** tab.

Project|Properties|Errors



There are three combo-boxes in **Project|Properties|Errors** and one edit box for the Error File. Complete the boxes as outlined on the next page.

- 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 CodeWright'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.

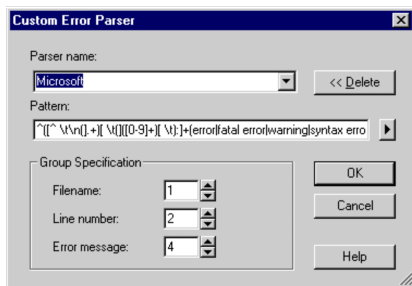
Note: Don't be too concerned if an error parser for your compiler isn't listed. When in doubt, try the Default parser (**DefaultErrorInfo**).

- The drop-down lists under the error parser combos contain the names of various error parsers available to CodeWright. More parsers can be added by marking the **Error Parsers** options in **Customize | Libraries**.
- Choose the appropriate error parser for the compiler being used. Usually the correct parser contains the compiler's name or the compiler's company name.
- 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 on *Custom Error Parsers*.

Custom Error Parsers

The error parsers available for CodeWright are listed in the drop-down lists under each of the three error parser combo boxes in **Project | Properties | Errors**. More error parsers can be added to the list by clicking the **Error Parsers** options in the list of **CodeWright 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, accessed by clicking the **Custom Error Parsers** button in the **Errors** dialog.

Custom Error Parser



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 are each made into separate entities.

A sample regular expression for making an error parser for the MS J++ Java compiler follows. Instructions for making the parser are also provided.

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, do 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|Properties|Errors** dialog.
5. Choose **_ErrorInfoParser MicrosoftJ++ Java** from the drop-down lists under the Error Parser combos.
6. Click **OK** in the **Project|Properties|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.

Navigating Build and Rebuild Command Output

The **Build** and **Rebuild** commands (on the **Project** menu) are treated differently from the **Compile** and **Debug** commands when it comes to error output navigation. In all cases the following occurs with error output in CodeWright:

- A list of errors displays in the **Build** tab of the Output Window.
- The error parser is applied to the error output file.
- The parser allows you to double-click on a line, or use the up and down arrow keys to highlight a line and hit , to access an error.
- After double-clicking or pressing , the appropriate piece of source code displays.

The following differences occur with the output produced by **Build** and **Rebuild** as opposed to the output produced by the **Compile** and **Debug** commands:

- A **Build** or **Rebuild** command just places the output in the Output Window, as described above.
- A **Compile** or **Debug Compile** command automatically positions the cursor on the correct line in the source file and places the error message text in CodeWright's status line when the source is open.

This behavior can be configured to your own taste. A CodeWright API call determines whether to show a list of errors or whether to jump directly to the first flagged file after a compile. These options are only available when using FTEE and the **Redirect Output** command option to capture and use output. Examples follow.

Example: To cause the error list to be displayed after each compile, add this line to the appropriate [Extension] section, or to the [Editor] section, of your CWRIGHT.INI file:

```
ExecShowCompileErrors=TRUE
```

There is an analogous call for **Build** and **Rebuild** commands. To have CodeWright to jump to the first error after such a command, add this line to the [Editor] section of CWRIGHT.INI:


```
ExecShowBuildErrors=FALSE
```

Note: For more information on **ExecShowCompileErrors** and **ExecShowBuildErrors**, see CodeWright's online help.















More information on FTEE can be found in the chapter *Set up a Compiler*. Information on the command options is available in this chapter under the topic *Command Options on the Tools Tab*.

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 **Build** toolbar to go to the next error .

Listed below are some keystrokes that belong to the CUA keymap that facilitate movement in the Output Window.

CUA Keystrokes	
Keystroke	Action
  	Bring up output window.
	Return to current document.
  	Parse next error.
 [Home]	Go to first line.
 [End]	Go to last line.
 	Go to next tab stop.
  	Go to previous tab stop.

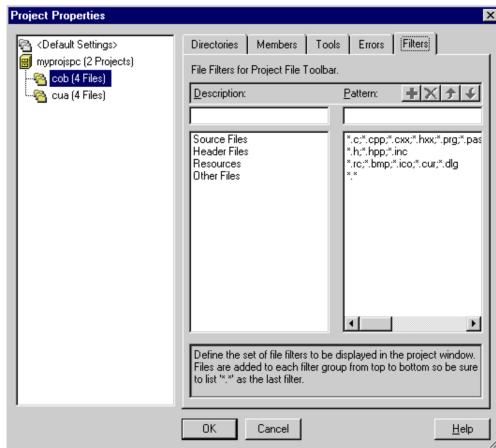
Filters tab of Project Properties

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*, in this chapter). The **Filters** tab controls how project files are "grouped" for display in the Project Window. The grouping is accomplished using wildcard characters and filename extensions.

To access/edit the appropriate version of the **Filters** tab, complete the following:

1. Select **Project|Properties**.
2. In the Project Properties List:
 - Select a project if the filters being viewed or modified are intended to be specific to the selected project only, OR
 - Select **<Default Settings>** if the filters being viewed or modified are intended to be global.
3. Click on the **Filters** tab.

Project|Properties|Filters



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:

1. Give the new filter a **Description** and a **Pattern**, for example "All Files" and *.*.
2. Click on **Add**. The new filter will be added to the top of the list.

Note: 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|Properties|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|Properties** 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|Properties|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.

2. The **Project|Properties|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.

Example: Your **Make** command might be

`${FTEE} nmake -f %y.MAK.` (where %y indicates the path and root of your project file)

If you already have a makefile, just name it explicitly instead of using %y. If you will be using VDOS instead of FTEE to watch the progress of the command, mark Use VDOS in the command options, and omit FTEE from the command.

Remember that the items on the **Tools** tab are only available if a project or the **<Default Settings>** item is chosen in the Project Properties List.

3. The **Project|Properties|Tools-Compile** category:

Select **Compile** from the drop-down list of tool categories under the **Category** combo box on the **Project|Properties|Tools** tab.

Compiler Command Lines: The compiler to use is first associated with the file type extension and then associated with the project. Make sure the right extension and project are selected. If you don't want the compiler to be associated with a project, choose **<Default Settings>** from the Project Properties List.

Using Project Spaces

Project spaces organize and house sets of projects. Once a project space is defined, its primary uses include:

- Selecting or changing projects.
- Selecting or changing project spaces.

Refer to the next topics.

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 checkmark 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.

Using Projects


Read on to find out about the following primary uses for a project that has been defined in a project space:

- Loading Files for Editing.
- Creating, Selecting and Saving Workspaces.
- Searching Project Files.
- Selecting files for version control Checkin or Checkout.

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|Properties** to add them (refer to the topic *Adding Files to a Project*).
- 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  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 *Characteristics of the File View Tab*). In addition, you may define filters (file specification patterns) to sort the file list into groups (see the topic *The Filters tab of Project Properties*).

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.

Note: 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.

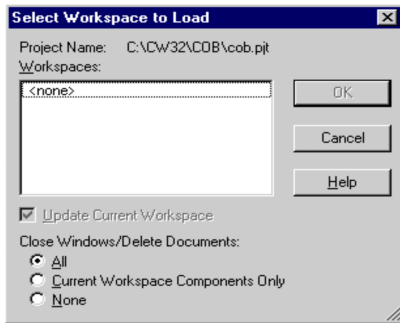
A workspace name may contain any printable characters (including spaces), but no apostrophe ('), backslash (\), quote ("), or square brackets ([]).

Creating a New Workspace

Suppose that you have a number of files open, and want to create a new workspace that doesn't include any of the files currently open:

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**. The **Select Workspace to Load** dialog displays.

Select Workspace to Load



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. CodeWright automatically updates the active workspace whenever you:

- Close or change projects.
- Exit CodeWright.
- (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 **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.
- 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 **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.

Searching Project Space and 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 detail in the Chapter on *Search and Replace and Navigational Tools*.

Selecting files for Check-in or Check-out

CodeWright's version control interface (**Checkin**, **Checkout** 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.

Project Files

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 CodeWright configuration file, you understand a CodeWright 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 CodeWright 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 CodeWright configuration or project file, keywords may be repeated. This is possible because CodeWright 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 the chapter on *Configuration Files & Command Line Parameters* of this manual.

Configuration and State Hierarchy

When you have no projects in use, CodeWright stores its configuration information in its configuration file (CWRIGHT.INI). Between sessions, CodeWright 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, and then setting options in the various tabs of the **Project|Properties** dialog. Various CodeWright dialogs also have options for specifying that dialog settings be stored with the project. Additionally, the **Directories** tab in the **Project| Properties** dialog has an option to **Store System Options in project file**. See the topic *Storing Configuration Settings with a Project*.

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 CodeWright.

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 CodeWright's main configuration files, see the chapter on *Configuration Files & Command Line Parameters*.

8- Set up a Compiler

CodeWright allows you to execute your compiles, assemblies, builds, etc. from within the editor. With proper configuration, the results of the operation will be placed in the **Build** tab on the Output Window. Double-click on the errors and warnings listed in the Output Window to bring up the source file with the cursor positioned at the line in which the error occurs.

All compiler configurations are done through the **Project|Properties** dialog. You do not have to use CodeWright projects in order to set up any compilers. However, if you do plan on creating and using projects, it is best to set up some default configurations (including compiler configurations) prior to creating the projects so that CodeWright will use them as defaults when additional projects are created. See the previous chapter on *Projects, Project Spaces, and Workspaces* for more information on CodeWright projects and setting project defaults.

Categories of Command Line Tools

As we have learned, CodeWright offers access to four kinds of command line tools in the **Project|Properties|Tools** dialog.

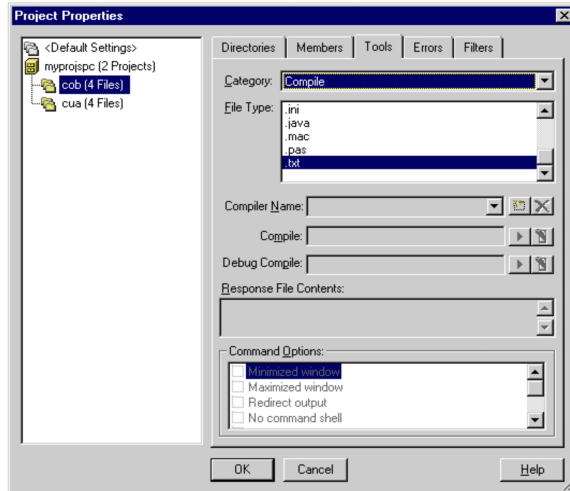
- The **Compile** command tools are tied to the filename extension as well as the project and act on the current open buffer. These commands are discussed in the next section.
- The **Build** and **Rebuild** command tools are not tied to the filename extension and an open buffer is not required. They send their output to an error file. After they run, CodeWright analyzes the error file to determine the location of errors that your compiler or linker has found.
- The **Custom** tools are run in whatever way you specify, and error parsers are not involved.
- The **VCS** tools are also run in any way specified, without involving error parsers.

Compiler commands include **Compile** and **Compile Debug**. The command lines are set up on the **Project|Properties|Tools** dialog under the **Compile** category of tools. To choose the **Compile** category, select it from the drop-down list under the **Category** combo.


Compiler Definition

Compiler commands are defined for the selected file type. Please ensure that you have the desired file type(s) selected from the list of file types before modifying settings

Project Properties Tools - Compile



The default compiler for the extension you selected is displayed in the **Compiler name** box:

- If the box is empty, or the wrong compiler is displayed, select the down arrow to the right of the box to display the list of other compiler names.
- If the compiler you seek is not listed, press the **New** button, type in a descriptive string for it, and enter an appropriate command.
- If the list is cluttered with compilers that you will never use, you can delete those you don't want (highlight the compiler and press .

The **Compiler name** field provides a way to save all of the information in this dialog under a name that can be used for later retrieval. If you fill out this dialog and set a compiler name, you can go back later to this same dialog, find the name in the **Compiler name** field, and instantly recall the settings you made.

Response File Contents

The **Response File Contents** edit box in the **Compile** category of tools is used for commands that may be too long or that may be better used in a batch file. It only comes into play if the %Q macro is being used on one or both of the compile command lines. See the topic *Response File* in the chapter *Projects, Project Spaces and Workspaces* for more information.

Compiler Command Line

A good starting point for the compiler fields on the **Tools** tab is to enter whatever would be typed at a DOS prompt for launching a compile.

Example: If you use Microsoft's Visual C++, a good general-purpose Compile command might be:

```
${FTEE} cl -DSTRICT -c -W3 -G3 -D_X86_=1 -DWIN32  
%r%e
```

The \${FTEE} is a command macro which calls the CodeWright FTEE.EXE program to capture the output of the compile. Refer to the topic *FTEE*, later in this chapter.

The %r and %e are Filename Component Macros that expand to the current file and extension. Refer to the listing of *Filename Component Macros* in the previous chapter on *Projects, Project Spaces, and Workspaces*.

The remainder of the above example is specific to each compiler.

Build Command Line

A good starting point for the **Build** field on the **Tools** tab is to enter whatever would be typed at a DOS prompt for the intended purpose. Filename component macros can be used as placeholders for specific file paths or path components, project names or directories. (Refer to the table under the topic *Filename Component Macros* in the previous chapter on *Projects, Project Spaces, and Workspaces* for a list of the macros.)

Example: If you use Microsoft's Visual C++, a good general-purpose **Make** command might be:

```
${FTEE} nmake -f %y.MAK
```

If the current project is *MYPROJ*, CodeWright will transform this line to

```
C:\CW32\FTEE.EXE nmake -f myproj.MAK
```

Other Tool Categories

The items listed in each respective category of the **Project|Properties|Tools** dialog (**Build, Rebuild, Compile** and so on) are all command lines. The same concepts apply when setting up these fields, as those used for setting up the **Compile** and **Debug** command lines.

Special Considerations

If you organize your files in a unique directory structure, there are some special considerations for the compiler to find the files needed.

As discussed in the topic *Working Directory* in the chapter *Projects, Project Spaces, and Workspaces*, CodeWright will shell out to DOS in the directory you have specified in **Project|Properties|Directories** to execute your commands. If you have designed a directory structure where your files are not all in the same directory, problems can arise.

Example: You might have a root application directory containing .exe's, a subdirectory containing object code, another subdirectory containing headers and include files, and another subdirectory containing your source files. Making the Compiler aware of where the necessary files for compiling are becomes tricky.

One solution would be to use a **Make** 'Description' file instead of the standard **Compile** line. To compile a specified target you might use the following command:

```
nmake -f MYPROJ.MAK %r.OBJ
```

This is basically a **Compile**, with the details supplied by the Description file rather than the command line or the environment.

Modifying the Command Line Environment

When setting up compilers in CodeWright it is usually necessary to set up custom environments, so that the operating system knows how to run the external command or utility. For example, it may be necessary to set the utility's directory using the PATH environment variable. These custom environments may include (or cause) the need for additional environment space. There are several ways in which to expand environment space:

- From the DOS command line, enter the following as the first thing on the line, prior to the compile or make command:

```
COMMAND.COM/E:4096/C
```

- In the Windows 95/98/2000/ME environments, create more environment space by modifying the CONFIG.SYS file and adding:

```
SHELL=command.com /p /e:4096
```

- In Windows NT 4.0, the environment can be manipulated on the **System Properties** dialog of the Windows NT Control Panel.

Using batch files instead of a compile or make commands is another solution.

- You might consider creating a batch file that sets up an environment common to all of your command lines. Prefix all of your normal command lines with this batch file, passing the program you want to run as parameters.

Example:

```
@ECHO OFF

REM Set up variables here...
SET WIN32=1
REM Now call the real program...
%1 %2 %3 %4 %5 %6 %7 %8 %9
```

Set up your build command line as: `${FTEE} BATCHFILE NMAKE`

- Another option is to set up a batch file that is unique to each command being run. For example, you might use `${FTEE} DOBUILD` in the command line, and in the DOBUILD batch file:

Example:

```
@ECHO OFF

REM Set up variables here...
SET WIN32=1
REM Now call the real program...
NMAKE
```

Displaying Output in CodeWright (FTEE and VDOS)

Most of the utilities that CodeWright's project tools will run will generate some sort of output. In most cases it will be necessary to use the output from within CodeWright (e.g. to access the errors generated by a compiler). There are two methods available for capturing and displaying output within CodeWright: FTEE and VDOS.

FTEE

If you want the output of your command to display in the DOS window while it is being redirected to the error output file, you can use our **FTEE** command:

Example: `${FTEE} nmake -f %Y.MAK`

The `${FTEE}` command macro calls the appropriate version of FTEE, a program that is located in the CodeWright directory. In the above example, the output of **nmake** will be split so that it goes to both the output file and the screen.

The FTEE utility (or FTEE32 for Windows NT 4.0), is supplied along with CodeWright. Just place the `${FTEE}` command macro, or the full path to FTEE.EXE, at the beginning of the command line. Since FTEE32 executes a command by creating a separate process for the command and capturing its output, it will only work for executable commands.

FTEE is similar to the several TEE programs found in the public domain and on many UNIX systems. Like a plumbing or electrical "Tee" connector, these programs allow output to be directed to two places at once. The difference between FTEE and these TEE programs is that FTEE allows the command to execute normally whenever you do not redirect output to a file. Other TEE programs typically quit with an error message under these circumstances.

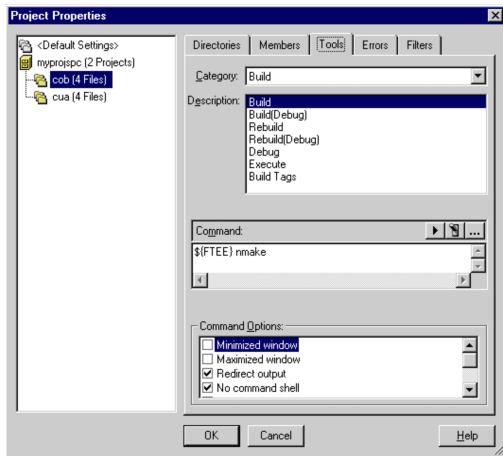
Before using FTEE, make sure that your application does not do any redirection on its own. Nothing useful can result from both your program and FTEE trying to redirect the same output.

Note: If you are calling a Windows executable, such as Borland's BCW, do not try to redirect output with FTEE. The program will be treated as a DOS application if you do.

Use VDOS

VDOS is a command shell that runs in a DOS window. Its output is automatically captured. When **Use VDOS** is checked for any of the tools in **Project|Properties|Tools**, VDOS will intercept output from **Compile**, **Build**, **Rebuild** and other project commands, and put it in the **Build** tab of the Output Window. Because it runs in the background, you can continue editing while your compile is executing.

Project|Properties|Tools



Remove references to other redirectors from your command lines when you use VDOS. Programs such as FTEE are not compatible with VDOS. Checking the **No Command Shell** box in **Project|Properties|Tools** for any of the project tools will disable the use of VDOS or FTEE for that command. For troubleshooting information, refer to the topic *VDOS Shell Window* (indexed under *VDOS*) in the online help.

Version Control Commands

We have learned from the previous and current chapters on projects and setting up compilers that in order to use various utilities in CodeWright, they need to be configured in the **Project|Properties|Tools** dialog. One of the tool categories in the dialog is for version control commands. The version control tools are one of the methods used for integrating version control in CodeWright. In all, there are two ways to use version control systems with CodeWright, but only one method can be used at a time. The methods are:

- To set up the tools with the appropriate command line version control utilities in **Project|Properties|Tools** under the **VCS** category.
- To use the SCC API (Source Code Control Application Programming Interface).

No matter which version control integration method is used, it is important to remember that CodeWright does not come with its own version control, but is configured to use external version control programs. To understand how to use version control in CodeWright, read the following chapter on *Version Control*.

9- Version Control

CodeWright does not come with its own version control system but it can be configured to integrate with an existing version control system. Once version control integration has been set up, there are a number of features in CodeWright that make performing version control operations easier.

This chapter is composed of three main topics. *Using Version Control in CodeWright* deals with the user interface for version control (i.e. menu items, windows, and popup menus, etc). The next two topics, *Using a Command Line Version Control Provider* and *CodeWright SCC Integration with Version Control Systems*, describe two alternative methods for CodeWright-version control integration.

Using Version Control in CodeWright

The items discussed in this section deal specifically with CodeWright's user-interface for version control (i.e. menu items, windows, and popup menus, etc). The information is based on the assumption that an existing version control system has been successfully integrated with CodeWright using one of the two methods described in the two ensuing topics on CodeWright-version control integration. Which integration method to use depends on the type of version control system available.

Version Control Menu

Most version control operations in CodeWright 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.

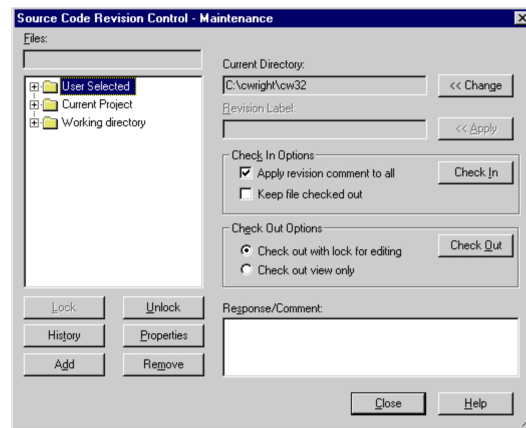
- Add or remove files from the version control project by clicking on **Add** or **Remove**.
- Access the Version Control Manager for the version control provider using the **Manager** item.

Source Code Revision Control - Maintenance

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. The **Source Code Revision Control Maintenance** dialog is a multi-purpose dialog that works for either the command line integration or the SCC Provider integration.

- ✓ This dialog is also available on the right-mouse-click popup menu in CodeWright’s Project Window (displayed by default on the left-hand side of the CodeWright window). If the Project Window doesn’t appear in CodeWright, click the **Project** menu item in the **Window** pull-down menu.

Source Code Revision Control Maintenance



The **Maintenance** dialog is useful for:

- 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.

Note: Any file opened in CodeWright 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 CodeWright project, although there are advantages to using CodeWright projects in conjunction with existing version control projects.

Version Control and CodeWright Projects

Refer to this section to:

- Associate Version Control projects with CodeWright projects.
- Add version control project filenames to your CodeWright project.
- Add CodeWright project files to an SCC Provider Project.

Associate Version Control Projects with CodeWright Projects

It is possible to associate version control projects with CodeWright projects when using SCC API version control integration (see the topic *CodeWright SCC Integration with Version Control Systems*). Doing so offers the following advantages:

- When a CodeWright project is opened, the associated version control project is also automatically opened, thus making version control facilities readily available to be used from within CodeWright. Checkin and Checkout (among other version control commands) are immediately available, allowing for quick preparation of files for editing.
- Closing the CodeWright project automatically closes the associated version control project.
- Files in the associated version control project can be “read” into the CodeWright project, providing a more convenient way to add appropriate files to the CodeWright project.

A version control project can be associated with a CodeWright project from one of two different dialogs: the **Tools | Version Control | Setup** dialog or the **Project | Properties | Members** dialog. To associate the two projects in the **Tools | Version Control | Setup** dialog, do the following:

1. Create a new project space with at least one member project (or open an existing one). For information about how to do this, see the chapter on *Projects, Project Spaces, and Workspaces*.
2. Set a current CodeWright 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 CodeWright project.
7. Verify that the working directory listed is that of the version control project's, not the CodeWright 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 the two projects in the **Project|Properties|Members** dialog, do the following:

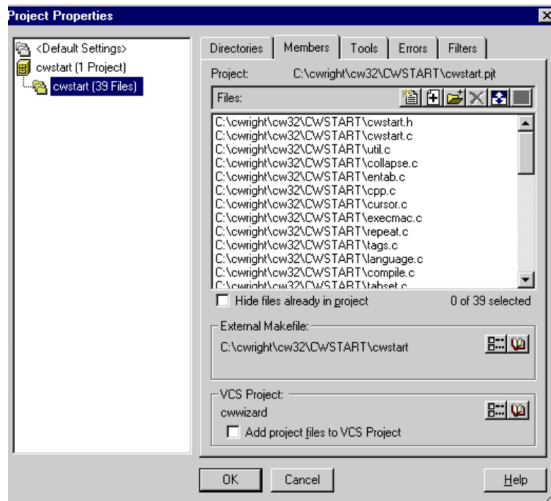
1. Assuming that a project space has been created with at least one member project, go to the **Project|Properties|Members** dialog.
2. Highlight the CodeWright project that you want to associate with the version control project.
3. In the **VCS Project** group, click the **Setup** button (the group 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 CodeWright 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 **OK** again.

Add Version Control Project Files to a CodeWright Project



Filenames that make up version control projects can easily be added to CodeWright projects when using the SCC service provider version control integration.

To add version control files to a CodeWright 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 CodeWright project. The version control **"Read"** feature can be found in the **Project | Properties | Members** dialog.

Project|Properties|Members



Address the following items that have to do with CodeWright's version control "Read" feature:

- **VCS Project:** - Lists the name of the current 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 listbox.






Having CodeWright automatically read the filenames of a version control project into a CodeWright project saves the time needed to manually add files, one by one, to a CodeWright project.


Add CodeWright Project Files to an SCC Provider Project

The **Project|Properties|Members** dialog also has a feature for adding filenames from a CodeWright project file to an SCC Provider Project:

1. Make an SCC Project association using the **Setup** button (as described in the preceding topic *Associate Version Control Projects with CodeWright Projects*).
2. Verify that a checkbox at the bottom of the tab becomes enabled, which will **Add Project Files to VCS Project**.
3. Click **OK** to close the **Project|Properties** dialog; each CodeWright 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.

CodeWright projects make version control more visual in CodeWright. When files are a part of a CodeWright project they will be displayed in CodeWright's Project Window in the **File View** tab. The icons that represent files are displayed differently depending on their status:

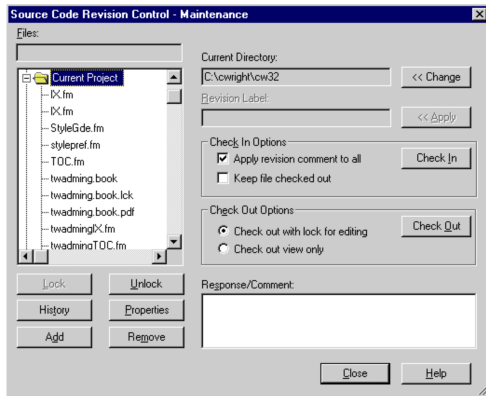
-  When the files are checked-in (or read-only) the icons that represent the files will be gray with horizontal lines in them.
-  When the files are checked out CodeWright conveniently places a red checkmark to the upper right of the file icons.
-  If another user has a lock on a file that is part of the project being worked on, CodeWright represents the file with a gray checkmark in the project window.
-  Some Version Control projects are set to delete work files when they are checked in. CodeWright 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, CodeWright will display the file's representative icon in a pale yellow color, indicating that it is an old version.

Note: 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.

Current Project Tree List

CodeWright 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 CodeWright 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.

Project Tree List



VCS and the User-Defined Popup Menu

CodeWright features a variety of customizable popup menus. 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 CodeWright's popup menu editor. The popup menu editor modifies menu description files that contain functions that form CodeWright popup menus. CodeWright'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 the chapter on *Custom 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. Here is how to add an item in the popup 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.

4. In the **Execute Function** field type
`MenuCommand IDM_TOOLS_VCS_GET.`
5. You can choose to have your new menu item inserted before the **Checkout** line by checking **Insert Before**.
6. Click **OK**, and then save the changes in the popup menu editor by clicking **File|Save**.
7. Click **OK** to close the dialog.

The right-mouse button popup menu should now contain three VCS commands.


Note: The above example works with command line or SCC VCS integration.

Making Your Own Version Control Popup Menu

A custom popup menu can be made in CodeWright with the popup menu editor. Use CodeWright'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 CodeWright's menu file, CWRIGHT.MNU.

The command to show a menu defined in CWRIGHT.MNU is **DlgMenuPopup**. Its syntax is described in CodeWright'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  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: Suppose you need access to two separate checkin commands without changing projects. You can set up each command with its own keystroke, button, or menu item.

1. The first thing to do is to create a section in your CWRIGHT.INI file for each checkin command. Here is a sample pair:

```
[VSCheckIn1]
CheckInSetCmd='put -cvcs1.cfg -M%Q %r%e'
CheckInBuffer
[VSCheckIn2]
CheckInSetCmd='put -cvcs2.cfg -M%Q %r%e'
CheckInBuffer
```

The CodeWright command **ConfigFileRead** can be used to execute the CodeWright commands contained in these sections.

2. To bind the first **checkin** command to a button or menu item, use this function:

```
ConfigFileRead='',VSCheckIn1
```

3. For the second checkin command, use

```
ConfigFileRead='',VSCheckIn2
```

If you're binding to a keystroke, use double quotes instead of the single quotes shown above.

The same method works for **checkout** commands. Sample CWRIGHT.INI file entries for checkout commands, with and without locks are displayed next:

```
[VSCheckOut1]
CheckOutSetCmd=0,'get -cvcs1.cfg %r%e'
CheckOutBuffer=0
[VSCheckOutLock1]
CheckOutSetCmd=1,'get -cvcs1.cfg -l %r%e'
CheckOutBuffer=1
[VSCheckOut2]
CheckOutSetCmd=0,'get -cvcs2.cfg %r%e'
CheckOutBuffer=0
[VSCheckOutLock2]
CheckOutSetCmd=1,'get -cvcs2.cfg -l %r%e'
CheckOutBuffer=1
```

Version Control Integration Configuration

CodeWright 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 (i.e. PVCS **GETEXE** and **PUTEXE**, or MS **SS.EXE**). The SCC Application Programming Integration is for integrating GUI version control systems with CodeWright's GUI.

The version control capabilities that CodeWright provides will 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 CodeWright'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 CodeWright. Choose the method that applies to the type of version control system you have (SCC for GUI, or Command Line for DOS).

Using a Command Line Version Control Provider

CodeWright 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 Merant's PVCS, and various versions of the RCS utility, ported from UNIX.

CodeWright integrates with command line version control utilities by shelling to DOS to run the utility and then returning to CodeWright once the deed is done. There are ways to cause the operation to be performed on the file that is currently open in CodeWright. It is also possible to have the operation performed on multiple files at once.

In order to use CodeWright's 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 CodeWright **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:

```
_StateHistory=XVCS,RCS,TLIB,PVCS,SourceSafe
```

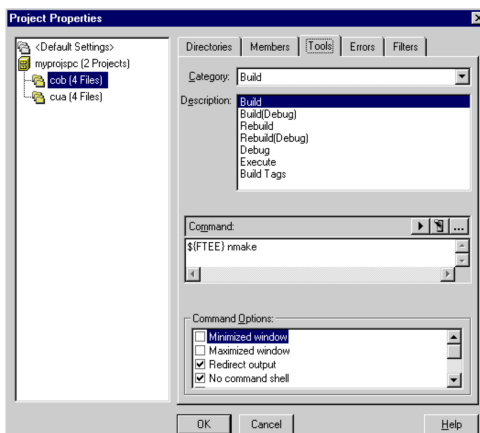
in the [Editor] section of CodeWright's configuration file, CWRIGHT.INI. The CWRIGHT.INI file can be found in CodeWright's home directory.

Customizing Version Control Commands


Whether you use existing version control command lines in CodeWright or make your own, you will need to pay a visit to the **Project | Properties** dialog if you want to define or modify them. Complete the following steps:

1. In the **Tools | Version Control | Setup** dialog, choose the **Configure** button. The **Project | Properties | Tools** dialog will be activated.

Project | Properties | Tools



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 will be executed whenever you select **Check In** from any of the various menus and toolbars that provide the **Check In** item. The command includes CodeWright filename component macros.
5. To see exactly how the command will run, select the **Test** button . Note that, when you have a dummy file such as FOO.C opened in CodeWright, 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|Properties|Tools** for version control are explained 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 will be used:

```
Put %b%e
```

- The initial **check-in** command defined in your CWRIGHT.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 CodeWright 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 CodeWright filename component macro that expands out the complete workfile name, less the extension.

%e is a CodeWright 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

Check-out Command

The **Checkout** 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 will be used:

Get %b%e

- If you 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 like the one below:

Get @%Q %b%e

- Use the **Checkout 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 **Checkout** command or leave it blank.

If you don't define a command for Checkout with locking, the following command will be 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 like the one below:

Get -l @Q %b%e

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|Properties|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 company's source code control operation standards.

Note: Predefined command lines are provided for several version control vendors. You select between these in the **Tools|Version Control|Setup** dialog.

At this point you should be completely set and ready to use DOS version control utilities in CodeWright. However, if any of the above steps didn't go as planned, the following explanations and tips may be of assistance.

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|Properties|Tools** dialog. The output will then be displayed in the **Build** tab of CodeWright's Output Window.

Note: By default, the Output Window is docked on the bottom edge of the CodeWright window. If it isn't visible, go to CodeWright's **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 CodeWright configuration file, CWRIGHT.INI. This line is typically placed in the [Editor] section in CWRIGHT.INI:

```
StateHistory=XVCS,RCS,TLIB,PVCS,SourceSafe
```

XVCS is the name of the list internal to CodeWright. This key is required. The names in the list follow the XVCS keyword on the line. Add to or edit this line as appropriate.

CodeWright SCC Integration with Version Control Systems

CodeWright provides a configurable interface for use with your Source Code Control System. CodeWright 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 CodeWright's SCC Provider support is based on the Microsoft Common Source Code Control (SCC) Specification. This spec describes a dozen or so entry-points that cover various version control operations (**checkin**, **checkout**, etc). 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 CodeWright can hook up their dialog boxes to these entry-points and get more or less the same functionality, independent of SCC Provider.

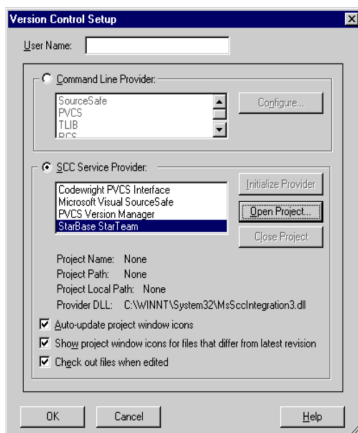
To find out whether a particular Version Control Provider offers Common Source Code Control (SCC) Integration, check with the vendor.

Version Control for Use with a Source Code Provider DLL

Make sure you have a properly installed Source Code Control System on your computer. Then complete the following steps to configure your SCC Provider:

1. Select **SCC Service Provider** in **Tools|Version Control|Setup**.

Version Control Setup



2. Select your version control system from the list in the dialog.
3. Press the **Initialize Provider** button in the **Version Control Setup** dialog.

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 CodeWright. If any of the above steps didn't go as planned, however, 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 CodeWright'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 integration modules don't even come on the same CD. The version control vendor should 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 CodeWright 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 CodeWright'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.

Note: The source code control vendor should make the appropriate system entries for SCC integration during the installation of the version control system. If the Source Code Control System is installed on a network server, 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.

Chapter 10

10- Synchronization

CodeWright's ground breaking effort to "synchronize" with alternate development environments continues. It now synchronizes with the following environments:

- Microsoft's Visual C++
- Microsoft's Developer Studio
- Microsoft's Visual Studio
- Borland's Delphi Development Environment
- Borland's C++
- Borland's C++ Builder Development Environment
- Visual Basic
- Texas Instruments Code Composer Studio

CodeWright's Sync Technology allows the stand-alone editor to be used along with development environments without concern for extra loading and reloading, or losing features or edits. Editing can be done in both environments with changes instantly updated in the alternate environment. In addition, keystrokes are available that will access menu commands in the alternate environment.

Initial CodeWright Setup

To use synchronization for any of the development environments listed above, the respective synchronization modules must have been chosen during the initial CodeWright installation. If the development environments for syncing were not chosen at that time, it will be necessary to do a custom CodeWright installation in order to install them.

Once the necessary sync modules have been installed, there are some setup procedures to follow for each individual development environment. The sections that follow provide that setup information.

CodeWright's Synchronization Wizard, and Loading CWSync.DLL

There are two procedures that can always be performed if problems arise during synchronization setup. These procedures are provided below, and are valid regardless of the development environment being used. Additional troubleshooting steps, specific to the individual development environment, are provided in respective sync topics in CodeWright's online help. Just search for the keyword 'synchronization' then look up the help topic for your environment.

If problems arise when setting up CodeWright synchronization for any of the supported environments, do the following:

- Run CodeWright's **Synchronization Wizard** from the **Wizard Choices** dialog, accessed by clicking **Configuration Wizards** on the **Help** menu. The Wizard will copy the files needed for the respective environment, and make the registry entries necessary for synchronization.

If an item in the list is grayed out, it means that the respective IDE was not chosen for synchronization during the CodeWright installation. It will be necessary to custom reinstall CodeWright in order to make those items available.

- Make sure that the **Synchronization** checkbox is enabled in CodeWright's **Customize | Libraries** dialog.

The **Synchronization** checkmark in the **Libraries** dialog causes CodeWright to load the selected library into memory. It also adds a line to the CWRIGHT.INI file so that the DLL will automatically load when CodeWright starts up.

The **Synchronization** option in the **Libraries** dialog places the following entry in CodeWright's configuration file, CWRIGHT.INI:

```
[LibPreLoad]
LibPreLoad=CWSYNC.DLL
```

Synchronization Setup From Within the Development Environments

After verifying that the CodeWright synchronization module has been loaded in **Customize | Libraries**, it will be necessary to configure the respective development environments to use that synchronization. The setup procedure for each environment is slightly different, and will be described individually in the next sections.

After setting up synchronization you may want to access menu items in the sync'd environment using keystrokes, buttons, or menu items in CodeWright. The last section of this chapter, *Accessing Menu Items via Synchronization*, talks about setting up keystrokes (and the like) in CodeWright to access menu items in the various synchronized development environments. The process for setting them up is similar for each environment so it is explained in its own section at the end of the chapter.

MSVC++ File Synchronization

The CodeWright synchronization setup with Microsoft Visual C++ requires a separate application called VCSync. VCSYNC.EXE provides a user interface to configure the file synchronization utility and it serves as a loader for VCSYNCIN.EXE. It runs continuously, trying to detect the presence of MSVC. Once it does, it invokes VCSYNCIN.EXE, the program that actually does the synchronization. VCSync provides file synchronization between CodeWright and 32-bit MSVC++ versions 5.x or 6.x.

VCSync Setup

Perform the following steps to run and setup VCSync:

1. Run VCSync. This can be done in one of two ways.
 - Run VCSYNC.EXE using the **CodeWright MSVC Sync** program item in the CodeWright program group. For Windows 9x/2000/ME and NT 4.0, the CodeWright program group is accessed by going to **Start|Programs|CodeWright**. Click the program item to invoke VCSync and run it minimized. As long as it is running, it will periodically check to see if MSVC++ has been invoked. When it has, it will start VCSYNCIN.EXE, a background task. OR
 - Run VCSYNC.EXE to invoke MSVC so that only one program invocation is necessary, rather than two. Just specify the MSVC application name as a parameter to VCSYNC.EXE using a **-u** option. The **-u** option will cause VCSYNC.EXE to invoke VCSYNCIN.EXE and then exit. The following steps describe how to do this:
 1. In Windows, right-click on the **Start** button, then click **Open**.
 2. Open the **Programs** folder.
 3. Open the **CodeWright** folder.

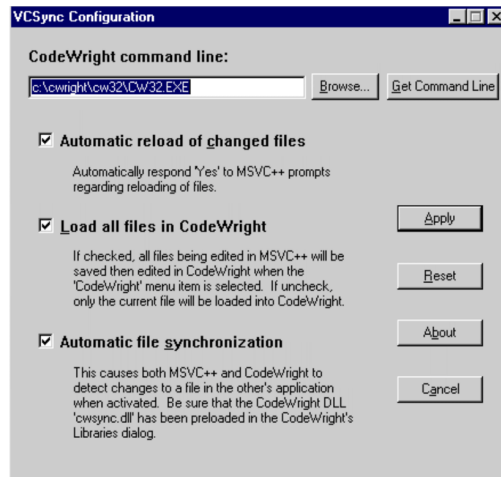
4. Right-click on the **CodeWright MSVC Sync** icon and click on **Properties**.
5. Click on the **Shortcut** tab and type something similar to the following in the **Target** field:

Example: VCSYNC.EXE -u "C:\Program Files\DevStudio\SharedIDE \bin\MSDEV.EXE"
(all on one line)

This command runs MSDEV.EXE, installs VCSYNCIN.EXE, and then closes VCSYNC.EXE.

VCSync initially runs minimized, as an icon in Windows 9x/2000/ME/NT 4.0's Systray. To make configurations in the following dialog, double click the icon to maximize the application.

VCSync Configuration



2. Perform the following steps to configure VCSync:
 - Enter the full path to the CodeWright executable (CW32.EXE) in the **CodeWright command line** edit field (if it is not already in your PATH), or use the **Get Command Line** button to automatically insert the command line for you.
 - The dialog's options provide runtime functionality between MSVC++ and CodeWright. They are set by default. Configure them as desired, according to the descriptions provided in the topic *Sync Configuration Options*.
3. After VCSync has been loaded and configured, re-minimize it.

4. Now go to MSDEV. In MSDEV, click **Tools|Customize...** and select **Addins and Macro Files**.
5. If the **CodeWright Add-in** is not already listed:
 - a. Select **Browse**. Set **Files of Type** to **Add-ins (.DLL)**. Add CWADDVC.DLL from the CodeWright home directory.
 - b. Open CWADDVC.DLL in your CodeWright home directory.
6. Check the selection box for **CodeWright MSDev 5.x (6.x) Synchronization** and close the **Customize** dialog.

Once the above steps are performed, a CodeWright toolbar button appears in MSVC. The button can be moved, docked, or attached to any visible toolbars.

A First View

Once you have set up VCSync appropriately, start Developer Studio (if it is not already running). There should be a new button. Click on this button to start CodeWright with all the same files that were loaded in Developer Studio's text editor. A switch between the two programs will automatically reload files that were changed in the other.

When synchronizing CodeWright and MSVC, changes to files are detected at three different times during normal editing:

- When either MSVC++ or CodeWright is first invoked.
- When either MSVC++ or CodeWright becomes the active application.
- When either MSVC++ or CodeWright edits an existing file on disk.

Read the topic *Bi-directional Synchronization*, later in this chapter, to find out about the **MSDevSync** toolbar that sync's backwards, from CodeWright to Developer Studio. Also see *Accessing Menu Items via Synchronization* to access Developer Studio menu items from CodeWright.

Delphi File Synchronization

The DPRSync utility manages file synchronization between Borland's Delphi and CodeWright. This utility requires Delphi 4.0 (32-bit) or newer and 32-bit CodeWright 4.0c or later. The utility consists of two files: DPRSYNC.BPL, a Delphi expert, and CWSYNC.DLL, a CodeWright Add-On DLL.

- For Delphi 4.0 users, the Delphi expert file is DPRSYNC4.BPL.
- For Delphi 5.0 users, the Delphi expert file is DPRSYNC5.BPL

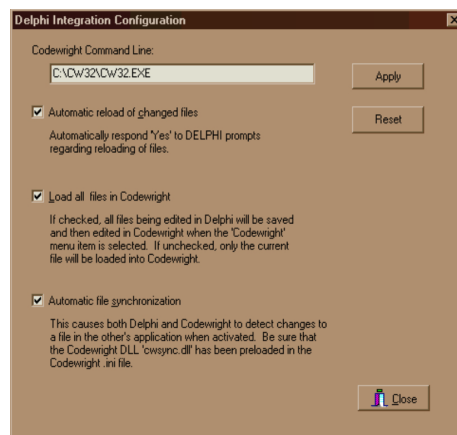
Note: If the correct options were selected during the CodeWright installation, the required modules will have been installed in their necessary locations for use by Delphi-CodeWright synchronization. If problems arise, see the *Troubleshooting* topic for Delphi Synchronization in CodeWright's online help.

DPRSync Setup

Perform the following steps to configure DPRSync:



1. Start the Borland Delphi IDE.
2. In the **Component|Install Package...** dialog, select the **Add** button.
3. If it is not already there, add the DPRSYNC4.BPL (DPRSYNC5.BPL for Delphi 5) file found in the CodeWright directory. You will see *CodeWright - Delphi Sync Package* show up in the list of loaded packages.
4. After completing the above steps, you will see two new menu items:
 - **CodeWright!** (found on the right side of the menu bar).
 - An **About CodeWright** item on the **Help** Menu that accesses a Configuration dialog for the Synchronization program.
5. From the Delphi **Help** menu, choose **About CodeWright**.
6. Select the **Settings** button to access the **Delphi Integration Configuration** dialog:

Delphi Integration Configuration





7. Enter the full path to the CodeWright executable (CW32.EXE) in the **CodeWright command line** edit field (if it is not already in your PATH), or use the **Get Command Line** button to automatically insert the command line for you.
8. The dialog's options provide runtime functionality between Delphi and CodeWright. They are set by default. Configure them as desired, according to the descriptions provided in the topic *Sync Configuration Options*.

A First View

Open a sample project in Delphi. Select **CodeWright!** from the Delphi main menu, or use the keyboard shortcut  . CodeWright should activate. The files that were open in Delphi should automatically open in CodeWright.

Note: Synchronization happens when going from Delphi to CodeWright by clicking on the **CodeWright!** menu item or by executing an

  shortcut. Synchronization going from CodeWright to Delphi happens when CodeWright loses focus and Delphi gains it.

Known Problems

- Automatic Generation of Event Handlers: One problem may occur immediately after returning to Delphi from editing a form in CodeWright. If the first operation you perform is to double click on a control to insert an event handler, the handler may not get inserted into the correct position within the form's source file. A workaround is to place the form into a modified state first. For example, insert a space, then delete the space.
- Forms inheritance: A problem exists when editing dependent forms in CodeWright. Once the child has been modified in CodeWright, Delphi will be unable to reload the parent until Delphi has been restarted.

Borland C++ File Synchronization

The BCWSync utility manages file synchronization between Borland's C++ and CodeWright. This utility requires 32 bit Borland C++ 5.0 and 32-bit CodeWright 5.0c or later.

The utility consists of four files:

- BCWSYNC.DLL, a Borland Add-On Expert
- BCWSYNC.SPP, a Borland Script file

- CWSYNC.DLL, a CodeWright Add-On DLL
- BCWADDON.DLL, an updated Borland DLL (use only if needed)

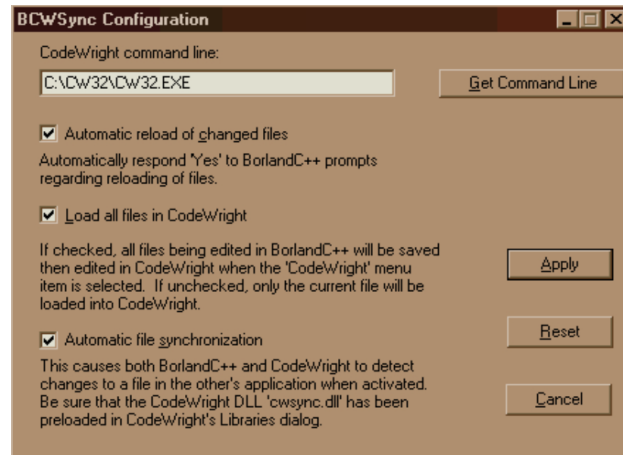
Note: If the correct options were selected during the CodeWright installation, the required modules will have been installed in their necessary locations for use by Borland C++-CodeWright synchronization. If problems arise, see the *Troubleshooting* topic for Borland C++ Synchronization in CodeWright's online help.

BCWSync Setup

Perform the following steps to configure BCWSync:

1. Start the Borland C++ IDE. You will see two new menu items:
 - **CodeWright!** (found on the right side of the menu bar).
 - An **About CodeWright** item on the **Help** Menu that accesses a Configuration dialog for the Synchronization program.
2. From the Borland C++ **Help** menu, choose **About CodeWright!**.
3. Select the **Settings** button to access the **BCWSync Configuration** dialog.



BCWSync Configuration



4. Enter the full path to the CodeWright executable (CW32.EXE) in the **CodeWright command line** edit field (if it is not already in your PATH), or use the **Get Command Line** button to automatically insert the command line for you.
5. The dialog's options provide runtime functionality between Borland C++ and CodeWright. They are set by default. Configure them as desired, according to the descriptions provided in the topic *Sync Configuration Options*.

A First View

Open a sample project into Borland C++. Select **CodeWright** from the IDE's main menu. CodeWright should activate.

Press   and the IDE will start a build if you made the key bindings described in the section on *Accessing Menu Items via Synchronization*.

Borland C++ Builder File Synchronization

The BCBSync utility will manage file synchronization between CodeWright and Borland's C++ Builder. This utility makes it easy to switch back and forth between the two applications, and makes sure that documents are kept up to date when you switch. This utility requires C++ Builder 4.0, or 5.0 and 32-bit CodeWright. The utility consists of two files: BCBSYNC4.BPL and CWSYNC.DLL.

- For C++ Builder 5.0 users, the name of the expert file is BCBSYNC5.BPL.

Note: If the correct options were selected during the CodeWright installation, the required modules will have been installed in their necessary locations for use by C++ Builder-CodeWright synchronization. If problems arise, see the *Troubleshooting* topic for C++ Builder Synchronization in CodeWright's online help.

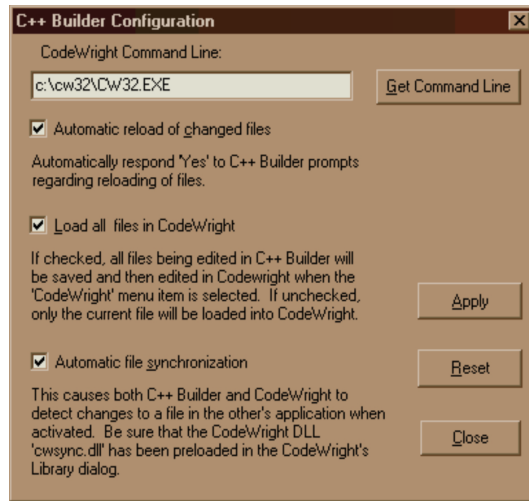
BCBSync Setup

Complete the following steps to setup synchronization for Borland's C++ Builder 4.0 or 5.0:

1. Start the Borland C++ Builder IDE, you will see two additional menu items in :
 - **CodeWright!** (found on the right side of the menu bar).
 - An **About CodeWright** item on the **Help** Menu that accesses a Configuration dialog for the Synchronization program.
2. Go to Builder's **Component | Install Package...** dialog.

3. Verify that the **CodeWright - C++ Builder x Sync Package** is loaded (where 'x' is the Builder version). If not, click **Add** and load BCBSYNCx.BPL from the CodeWright directory.
4. Click **OK**.
5. From the Builder **Help** menu, choose **About CodeWright**.
6. Select the **Settings** button to access the C++ **Builder Configuration** dialog:

C++ Builder Configuration




7. Enter the full path to the CodeWright executable (CW32.EXE) in the **CodeWright command line** edit field (if it is not already in your PATH), or use the **Get Command Line** button to automatically insert the command line for you.
8. The dialog's options provide runtime functionality between Borland C++ Builder and CodeWright. They are set by default. Configure them as desired, according to the descriptions provided in the topic *Sync Configuration Options*.

A First View

Open a sample project into Builder. Select **CodeWright** from the Builder main menu, or use the keyboard shortcut **ALT + I**. CodeWright should activate.

Press **CTRL + P**. The **Builder Project Info** will display, if you made the key bindings described in the section *Accessing Menu Items via Synchronization*.

Notes:

- Synchronization happens when going from Builder to CodeWright by clicking on the CodeWright menu item or by executing an  shortcut.
- Synchronization going from CodeWright to Builder happens when CodeWright loses focus and Builder gains it.
- This synchronization can only be reliably achieved if the Builder BCBSYNC.DLL expert file is loaded as described above and the CodeWright CWSYNC.DLL file was loaded during CodeWright installation.
- When synchronizing Builder units, the .CPP file and the .H file are both opened. Cursor positions are only set on the .CPP file. The .H file's cursor position will remain in the position CodeWright had it last. When changes are made to the Builder's Open Tools interface to access the .H file, an update will be made to sync the cursor position for the unit's .H files.

Visual Basic File Synchronization

The VBSync utility manages file synchronization between CodeWright and Visual Basic 6.0. This utility makes it easy to switch back and forth between the two applications, and makes sure that documents are kept up to date when you switch. This utility requires Visual Basic 6.0 and 32-bit CodeWright. The VBSYNC utility consists of 4 files:

- VBSYNC.DLL: A Visual Basic 6.0 Add-in, Syncing VB to CW
- CWVBSYNC.DLL: A DLL called by VBSync.DLL
- CWSYNC.DLL: CodeWright Add-On DLL, Syncing from CW to VB
- BAS.DLL: Updated version of the CodeWright Add-on for Basic language support. The DLL now offers language support for the following file types: .CLS, .DSR, .DOB, .FRM, .CTL, and .PAG.

This program will NOT sync components with the following extensions:

.FRX,.DOX,.RES

These extensions will be saved when the CodeWright toolbar button is clicked, but they will not be opened up into CodeWright. These files are in a proprietary binary format and are not good candidates for editing in CodeWright as text files.

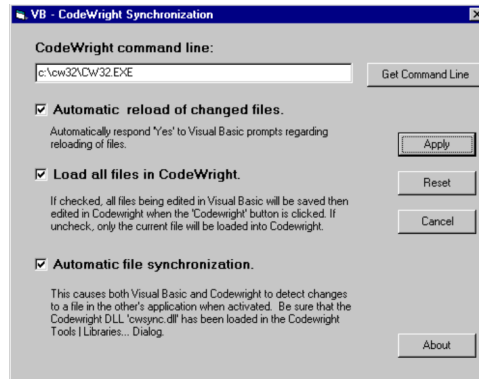
Note: If the correct options were selected during the CodeWright installation, the required modules will have been installed in their necessary locations for use by Visual Basic-CodeWright synchronization. If problems arise, see the *Troubleshooting* topic for Visual Basic Synchronization in CodeWright's online help.

VBSync Setup

Do the following to setup Visual Basic Synchronization with CodeWright:

1. Start the Visual Basic IDE. You will see three additional menu items:
 - A **CodeWright - VB Sync** toolbar button to SYNC between VB 6.0 and CodeWright.
 - The VBSync command menu item, **CodeWright!**, in the **Add-Ins** menu. It does the same thing that the toolbar button does.
 - The VBSync configuration menu item **CodeWright Sync...** in the **Add-Ins** menu.
2. From the **Add-Ins** menu, choose **CodeWright Sync...** to access the **VB-CodeWright Synchronization** dialog:

VB-CodeWright Synchronization



3. Enter the full path to the CodeWright executable (CW32.EXE) in the **CodeWright command line** edit field (if it is not already in your PATH), or use the **Get Command Line** button to automatically insert the command line for you.
4. The dialog's options provide runtime functionality between Visual Basic and CodeWright. They are set by default. Configure them as desired, according to the descriptions provided in the topic *Sync Configuration Options*.

A First View

Open a sample project into Visual Basic 6.0. Select the **CodeWright** toolbar button. CodeWright should activate and any files loaded in Visual Basic will subsequently be loaded in CodeWright.

- Synchronization happens when going from VB60 to CodeWright by clicking on the **CodeWright - VB Sync** toolbar button or the **CodeWright!** menu item. Synchronization going from CodeWright to VB60 happens when CodeWright loses focus and VB60 gains it.
- This synchronization can only be achieved reliably if the VBSYNC.DLL Add-In and the CodeWright CWSYNC.DLL Add-On files are both loaded as described above.

TI Code Composer Studio File Synchronization

The TICCSync utility manages file synchronization between CodeWright and Texas Instruments Code Composer Studio. It only works with Code Composer Studio (TM) from Texas Instruments. The utility makes it easy to switch back and forth between the two applications, and makes sure that documents are kept up to date when you switch. It requires TI Code Composer Studio (v1.x) and 32-bit CodeWright 6.5 or later.

The TICCSYNC utility consists of 3 files:

- CWSYNC.DLL: A CodeWright Add-On DLL that provides the sync from CodeWright to TI Code Composer Studio.
- TICCSYNC.DLL - A TI Code Composer Studio plug-in that synchronizes TI Code Composer Studio with CodeWright.
- TICCSYNC.EXE - A stand-alone program that provides automatic loading of the synchronization utility and a user interface for its configuration.

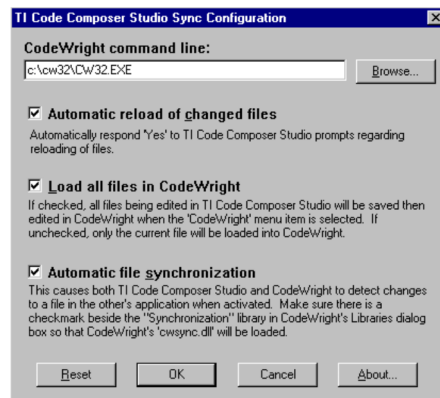
Note: If the correct options were selected during the CodeWright installation, the required modules will have been installed in their necessary locations for use by TI Code Composer Studio-CodeWright synchronization. If problems arise, see the *Troubleshooting* topic for TI Code Composer Synchronization in CodeWright's online help.

TICCSync Setup

Do the following to setup TI Code Composer Studio synchronization with CodeWright:

1. Start the TI Code Composer Studio IDE. You will see two additional menu items:
 - A **CodeWright!** menu item on the **Tools** menu in Code Composer Studio. Select it to launch CodeWright and synchronize files.
 - A **CodeWright Settings...** menu item on the **Tools** menu in Code Composer Studio. Select it to launch a sync configuration dialog box.
2. From the **Tools** menu, choose **CodeWright Settings...** to access the **TI Code Composer Studio Sync Configuration** dialog:

Code Composer Sync Configuration



3. The dialog's options provide runtime functionality between TI Code Composer Studio and CodeWright. They are set by default. Configure them as desired, according to the descriptions provided in the topic *Sync Configuration Options*.
4. Activate the TICCSYNC.DLL plug-in (this step may have already been completed from actions performed in step 2). To do this either:
 - Select **CodeWright!** or **CodeWright Settings...** from the Code Composer **Tools** menu. Once started the TICCSYNC plug-in will remain active until Code Composer Studio shuts down. OR
 - Run TICCSYNC.EXE from the CodeWright home directory. The program runs minimized in the operating system's Systray. TICCSYNC.EXE periodically attempts to detect the presence of Code Composer Studio. When detected, TICCSYNC.EXE activates the TICCSYNC.DLL plug-in. TICCSYNC.EXE runs continuously in the Systray until it is turned off.

A First View

Once you have set TICCsync up appropriately, start Code Composer Studio. Select **CodeWright!** from the **Tools** menu. CodeWright will be activated with all the same files loaded that were loaded in Code Composer Studio's text editor. When you switch tasks between the two programs, each will automatically reload any source files that were changed in the other, with the cursor position maintained. Changes to files are detected at three different times during normal editing:

- When either Code Composer or CodeWright is first invoked.
- When either Code Composer or CodeWright becomes the active application.
- When either Code Composer or CodeWright edits a file on disk.

Notes:

- Synchronization happens when going from Code Composer to CodeWright by clicking on the **CodeWright!** menu item.
- This synchronization can only be achieved reliably if TICCsync.DLL and CodeWright CWSync.DLL are both loaded as described above.

Read the following topic *Bi-directional Synchronization* to find out about the **TICCsync** toolbar that sync's backwards, from CodeWright to Code Composer Studio. Also see *Accessing Menu Items via Synchronization* to access Code Composer Studio menu items from CodeWright.

Bi-directional Synchronization

The synchronization modules for Visual Studio 98 and TI Code Composer Studio each have toolbars that make VCSync and TICCsync bi-directional. This means that, when the appropriate button is pressed, files in CodeWright will automatically load in Developer Studio or Code Composer Studio, with the cursor position intact. The **MSDevSync** and **TICCsync** toolbars can be enabled in CodeWright's **Customize | Toolbars** dialog. They have buttons for syncing either the current file or all files open in CodeWright with the other development environment. The toolbars can also set breakpoints in Visual Studio, one at a time, and it can build and compile the appropriate files for both Visual Studio and Code Composer Studio.

Sync Configuration Options

The following configuration options are available in CodeWright Sync Configuration dialogs. Use them according to the descriptions provided.

- The **CodeWright Command line** edit box is for the location (path) and command for CW32.EXE. The full, correct directory path and name of the CodeWright executable are needed if CW32.EXE is not set in the PATH environment.

- The **Get Command Line** button retrieves the path to the CodeWright executable file (CW32.EXE) from the registration database and inserts it in the **CodeWright Command Line** edit box.
- The **Browse** button accesses a dialog that allows you to browse for the location of the CodeWright executable (CW32.EXE). The file selected from this dialog will be inserted in the **CodeWright command line** box.
- The **Automatic reload of changed files** check box causes an automatic 'Yes' response to occur for any prompts regarding file reloading in the sync'd environment. This will override dialog boxes that warn if files have been changed or overwritten externally.
- The **Load all files in CodeWright** check box causes all files being edited in the sync'd environment, along with their line and column numbers, to be sent to CodeWright for editing. The operation occurs after an automatic **File|Save All** has occurred in the sync'd environment. If this box is NOT checked, a **File|Save** will only be performed on the current file in the sync'd environment. Then the file and its line and column numbers, will be sent to CodeWright.
- The **Automatic file synchronization** check box causes both the sync'd environment and CodeWright to detect changes to a file in the other's application. If checked, changes to files in one application will be detected, but the user will NOT be prompted to save them before reloading the file in the other application. The save and reload will be done automatically. If NOT checked, changes to a file in one application will be detected, but the user will be prompted to save the file before reloading it in the other application.
- The **Reset** button resets the original settings to those in effect when you entered the dialog.
- The **Cancel** button cancels out of the dialog without saving the changed settings.
- The **OK** or **Apply** buttons save changes to the registration database.
- The **About** button displays an **About** dialog containing version information.

Note that the behaviors of the settings in the Configuration dialogs only occur when the **CodeWright!** menu item is invoked from the sync'd environment or when switching between CodeWright and the sync'd environment using any of the designated key combinations.

Accessing Menu Items via Synchronization

Each of the synchronization utilities described in the previous sections have functions that allow menu commands for the respective synchronized environment to be called from within CodeWright. This allows tasks that are commonly performed in the development environment, (e.g. builds and compiles) to be performed from within CodeWright.

For example, the following function accesses the **Open** option on the **File** menu in Microsoft Developer Studio:

```
cwsyncvc_menustr 'FileOpen'
```

Functions such as the one above have been configured in different sections of a CodeWright configuration file called CWSYNC.INI, located in the CodeWright home directory. As stated, the functions are intended to provide more convenient access to commonly used menu items in the sync'd environment.

The sections in CWSYNC.INI can be accessed by uncommenting some entries in the [KmapAssign] section of CodeWright's main configuration file CWRIGHT.INI (also located in CodeWright's installation directory). The entries in CWRIGHT.INI appear as follows:

```
; Uncomment for Visual Studio 97 (MSDev 5.0 or MSDev 6.0) Menu commands:
;ConfigFileRead( CWSYNC.INI, KmapAssign_MSVC50, FALSE )

; Uncomment for VBSync MS Visual Basic 6.0 Menu commands:
;ConfigFileRead( CWSYNC.INI, KmapAssign_VB60, FALSE )

; Uncomment for CWSync Delphi 5.0 Menu commands:
;ConfigFileRead( CWSYNC.INI, KmapAssign_DPR50, FALSE )

; Uncomment for CWSync Delphi 4.0 Menu commands:
;ConfigFileRead( CWSYNC.INI, KmapAssign_DPR40, FALSE )

; Uncomment for CWSync Borland C++ Builder 4.0 Menu commands:
;ConfigFileRead( CWSYNC.INI, KmapAssign_BCB40, FALSE )

; Uncomment for CWSync Borland C++ Builder 5.0 Menu commands:
;ConfigFileRead( CWSYNC.INI, KmapAssign_BCB50, FALSE )

; Uncomment for CWSync Borland C++ 5.0 Menu commands:
;ConfigFileRead( CWSYNC.INI, KmapAssign_BCW50, FALSE )

; Uncomment for CWSync TI Code Composer Menu commands:
;ConfigFileRead( CWSYNC.INI, KmapAssign_TICClx, FALSE )
```

Each of the preceding entries corresponds to a separate section in CWSYNC.INI that lists key assignments for the appropriate synchronized environment. To enable the appropriate keystrokes, uncomment the entry in CWRIGHT.INI that matches the development environment being used. Remove the semicolon at the beginning of the line to uncomment an entry.

Example: To set up the keystrokes for Delphi 4.0 menu commands, uncomment the line that looks like:

```
;ConfigFileRead( CWSYNC.INI, KmapAssign_DPR40,  
FALSE )
```

See the topic *Assigning Menu Items to Keystrokes* under the *Synchronization* topic in CodeWright's online help for a list of key assignments in CWSYNC.INI.

11- Search and Replace and Navigational Tools

This chapter covers the many tools CodeWright has for navigating code. It starts by talking about CodeWright's unsurpassed **Search** and **Replace** features, including a discussion of **regular expressions**. It goes on to describe **Tags** and **Symbols**, CodeWright's browsing tools, and ends by talking about **Bookmarks** and **Button Links**.

Search and Replace and Regular Expressions

CodeWright offers a relatively large number of methods for performing search and replace operations. Each has its own advantages under differing circumstances. Some work on a single file and others work directly on the files on disk. Some are meant to be quick and simple while others are designed for power. To help you review your alternatives, the description of these methods are collected together in this section.

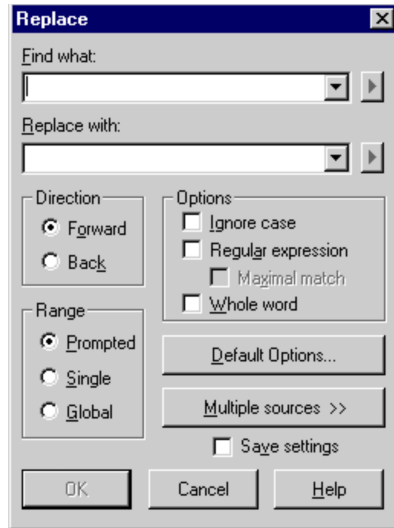
The topics covered in this chapter regarding search and replace are listed below:

- Search and Replace dialog
- Incremental Search
- Toolbar Search
- Quick Search
- Multi-source Search and Replace
- Search Options dialog
- Stop a long search
- Regular Expressions

Search and Replace Dialog

Many of the search features can be accessed from the **Search** menu. Clicking on 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. Refer to the following picture of the (Search and) **Replace** dialog:

Replace



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.

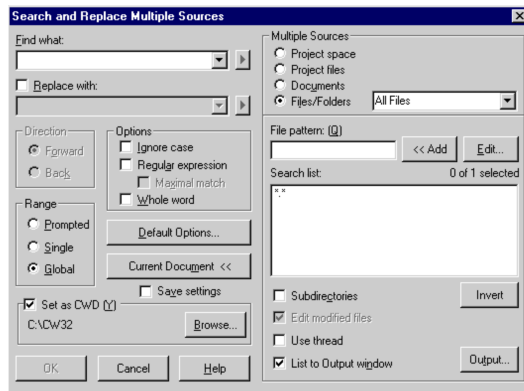
Save Settings

The **Save Settings** check box is provided to save you from editing the **Search Options** dialog whenever 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.


Search and Replace Multiple Sources



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 actually performing a **File Grep**. This 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 **Search** tab also displays the line numbers and the line containing the matching text.
- To move to any of the matches listed in the **Search** tab, double-click on it with the mouse, or select the matching line from the list and press .

Several features exclusive to the **Search and Replace Multiple Sources** dialog make it one of the more convenient navigating tools in CodeWright. 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 checkbox labeled **Set as CWD**. Items to consider with regard to the current directory are:

- **The Browse Button:** If you are not in the desired directory, press the **Browse** button to select a new working directory for CodeWright.
- **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

CodeWright 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 only:** 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.

- Note:** 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 CodeWright.
- 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 listbox 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 on 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 the  button and clicking on those members of the list.
- If the list of documents you *don't* 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 CodeWright 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 CodeWright'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 described next.

Search Pattern

For the **Edit File/Folders** dialog's **Search pattern** edit box:

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** listbox 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 listbox and the **Directories** tree list in the **Edit Files/Folders** dialog allow you to select the path you want to apply to the search set. This is only meaningful if the **Include Directory** checkbox 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:

- Use **Add**, or **Delete** to add or remove members of the **Search Patterns** list.
- Use **Invert** to reverse the current selection; those items that were selected become deselected and vice versa.
- Use **Clear** to deselect all the patterns on the list so that you can start selecting from scratch.

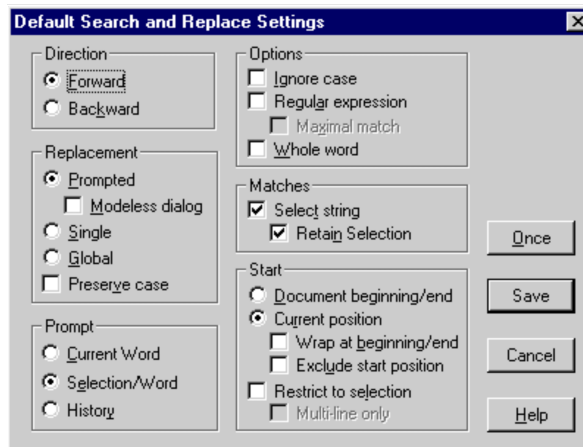
Default Button

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.

Default Search and Replace Settings



To access the **Default Search and Replace Settings** dialog, do one of the following:

- Click **Options** on the **Search** menu.
- Click **Default Options** in the **Search** dialog.
- Click **Default Options** in the **Replace** dialog.
- 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.

Replacement

The **Replacement** group of options in the **Default Search and Replace Settings** dialog is specific to the replacement operation. They are listed next.

- **Prompted replacement.** When the **Prompted** radio button is selected, you will be 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 makes available the **Modeless dialog** option.
 - ✓ **Modeless dialog.** This option works with single documents only. It works in conjunction with a prompted replacement. When CodeWright prompts with options to **Replace Current**, **Find Next**, **Replace Global**, etc, 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 lowercase.
 - ✓ 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, uses the word at the cursor.
- **History:** Uses the most recent response in the history list.

Search Options

The **Options** group of check boxes is also available in the **Search**, the **Replace**, the **Default Search and Replace Settings**, the **Search and Replace Multiple Sources**, and the **File Grep** dialogs. The options allow you to turn various search attributes on or off. They include:

- **Ignore case:** When 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, the search pattern is viewed as 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 CodeWright 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 CodeWright's **Multiple Source Search** dialog can be used to save valuable time follows.


Suppose you want to look for the string *version* in two directories, C:\DOCS*.TXT and D:\PROJECT*.C. Here's what to do:

1. Select **Search | Multiple Sources** to get to the **Search and Replace Multiple Sources** 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 on the **Edit** button to bring up the **Edit Files/Folders** 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 will appear 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 on the **Invert** button to highlight the list, or select just the specific files you wish to search.
10. Click on the **OK** button. The Output Window will appear, and you will see results of the search as the search pattern is found in the target files.

Note: The **List to Output Window** box has no effect on search-only operations. The results will appear regardless. 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.

✓ Often it becomes necessary to stop a long search. To do this, press -Break.

Fast Find on Standard Right-Click Popup




The standard right-click popup menu has a **Find** item for performing quick multi-source searches. The menu item only appears when a right click is performed while the document cursor is positioned at the beginning, middle, or end of a word. The search will be performed on that word. To use Fast Find, do the following:

1. Right-click in the current document on a word to be searched for.
2. Select the **Find <word> in** item in the popup menu. A submenu will appear.
3. From the submenu, select one of the following sets of documents.
 - **Files/Folders** (searches all files in the current directory)
 - **Documents** (searches all documents that are open in CodeWright)
 - **Project Files** (searches all files in the current project, if one is open)
 - **Project Space** (searches all files in the current project space, if one is open)
4. Use the **Search** tab on CodeWright's Output Window to see the search results.

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.

CodeWright's incremental search function is called **ISearch**. **ISearch** 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.

Begin an incremental search by invoking **ISearch**. This can be done from **Tools|API Command**, or by pressing    in the CUA keymap. **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 find you have mistyped, 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 . 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    key combination in the CUA keymap. There is also a **Quick Search** option in the right-click popup 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 listbox 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 , the search commences.

If CodeWright finds a match, the Toolbar Search will retain the focus. This gives you the opportunity to search again just by pressing  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, causing the history list to drop down.

Regular Expressions

Regular Expressions are powerful notations for matching string patterns. CodeWright makes UNIX-Style Regular Expressions available for its search feature. CodeWright also uses Regular Expressions for interactively customizing CodeWright's Symbols, embedded language configurations, user defined keywords parser, makefile reader, and Visual Studio workspace reader.

Note: 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 the chapter on *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 CodeWright'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. When reading the descriptions that follow, remember that 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 below:

Character	Meaning
.	Matches any single character, except a new line
*	Matches zero or more occurrences 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.

Character	Meaning
\	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 when immediately followed by a newline character; `cat\$` matches the word `cat` only when immediately followed by the character \$.

Below is a list of other escape sequences supported by CodeWright's 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. (e.g. 'cat')
- Ambiguously, by using the dot (.) metacharacter, if matching a character literally is too limiting. (e.g. '..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. (e.g. [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 character class:

```
[AEIOUYaeiouy]
```

matches any vowel, whether upper or lowercase.

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 [0-9] will match any character between 0 and 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 don't 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 can't 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 proceeding 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 are provided next.

Examples:

- `^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.

Example:

- `PUBLIC|PRIVATE` matches either the word "PUBLIC" or "PRIVATE".
- `PUBLIC(void|DWORD)` matches the word "PUBLIC", when followed by a single space, and then followed by either the word "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.

CodeWright 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, CodeWright 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 CodeWright'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 following examples are intended to inspire your own use of regular expressions:

Regular Expression Examples	
Pattern	Description
[A-Za-z_][A-Za-z0-9_]*	C language identifier
-?([0-9]+\.[0-9]* \.[0-9]+)	Floating point (real) number
([\t] ^)[^ \t]+	Beginning of word (simple white space)
[^ \t]+\c([\t] \$)	End of word (simple white space)
([^A-Za-z0-9] ^)[A-Za-z0-9]+	Beginning of word (Non-alphanumeric)
[A-Za-z0-9]+\c([A-Za-z0-9] \$)	End of word (Non-alphanumeric)

Regular Expression Examples	
Pattern	Description
/*.**/	Single-line comment (C language)
/*.*\n([\t]**.*\n)*.**/	Multi-line comment (C language)

Searching for Spaces, Tabs and other Blank Characters

One of CodeWright'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** menu 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. Mark **Regular Expression** and **Save Settings**.
4. Click **OK**.

Searching for New lines

To find new-line characters, complete the following steps:

1. Go to the **Search** pull-down and click on **Search**.
2. Type \n in the **Find What** field.
3. Mark **Regular Expression** and **Save Settings**.
4. Click **OK**.

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.

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 this string:

```
\x0C
```

(That's *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 this:

```
\x0C\x0D
```

Searching for New lines: Issues

Generally, `\n` is the suggested method for matching or replacing line ends in CodeWright, as described in the previous topic on *Searching for New lines* (the `\` must be doubled when used in code). In some cases, however, it is more desirable to search for new lines using their respective hexadecimal notations (0D0A, 0A, 0D, etc). 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:

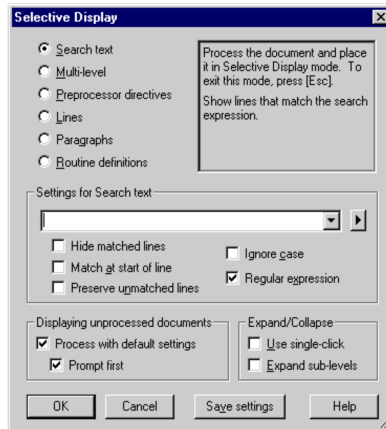
- `\x0A` will match a solo (without a preceeding 0x0D) 0x0A but not the pair.
- `\x0D` will match a solo (without a following 0x0A) 0x0D but not the pair.
- `\n` will match a solo 0x0A or a 0x0D, 0x0A pair.
- Replacing `"\x0D"` with `"\n"` would be good for converting a Macintosh file (opened with auto-detect 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 | Options**. More information on setting and using UNIX EOLs can be found in the chapter *UNIX*.

Selective Display




CodeWright's **Text** menu has an option for **Selective Display**. **Selective Display** is a feature that "collapses" or "folds" text in the current document so that only desired text is displayed or hidden. The **Selective Display** menu item accesses a dialog that has options for controlling which lines of the document are visible.

Selective Display



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,    also controls Routines selective display.

Note: 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 that find their source files cluttered with `#if`defs. 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 `#if`defs 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. `#if`defs are used to maintain different versions in a single file. Then, changes to common code only needs 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. CodeWright allows full benefit from the use of pre-processor conditionals by letting the code be seen more clearly for what is being worked on. Thanks to Selective Display Mode, the whole file is still there, but the parts that don't apply can be hidden.

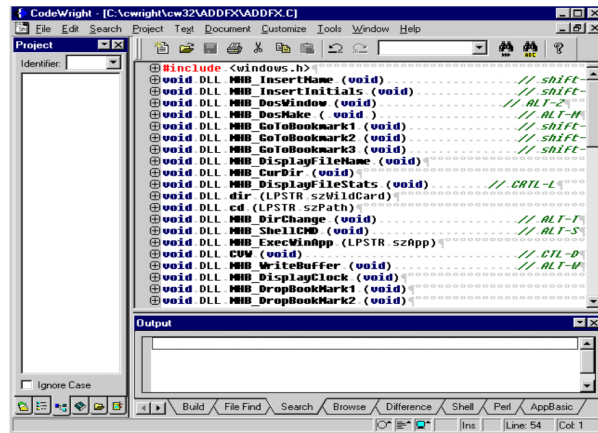
Adding "defines" to the **Preprocessor directives- Defined Constants** field allows specified defines to remain visible in preprocessed view. Multiple "defines" can be specified by separating them with semi colons. Once the necessary "defines" have been added, and desired options have been chosen, clicking the dialog's **OK** button will hide specified preprocess code.

Viewing/Hiding Lines

Once a document is in **selective** mode, 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   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   again.
- Double-clicking with the right mouse button also expands and collapses hidden text.

Selective Mode



Press **[ESC]** to restore the invisible lines. **[CTRL][SHIFT][C]** restores **Selective Display** for the previously selected lines in the CUA or BRIEF keymaps. **[F3]** restores **Selective Display** for the previously selected lines in the vi emulation. CodeWright will save the current selective display view between sessions if the option **Visible Lines** is marked in **Customize|Environment|State**.

Browse, Tags, Symbols and Objects

Whether working on someone else's code, with a long line of inheritance, or working on code you wrote last week, you can spend a lot of time navigating it. CodeWright provides the following text and source code tools for finding and navigating code:

- **Browser**
- **Tags**
- **Outline Symbols**
- **Objects Window**

Each of the above tools has advantages and disadvantages when used for navigational purposes. The next few paragraphs describe reasons why one might choose one over another.

Which Navigational Tool Should I Use?

Choose from the navigation methods described in the following topics, *Browser*, *Tags*, and *Outline Symbols*, or use them all.

Browser

The CodeWright Browser uses Microsoft .BSC files that are produced when compiling source code in Microsoft Developer Studio. It also uses compiled tags databases produced by the TAGSWnn utility shipped with CodeWright. 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. If you are compiling with Microsoft tools this method gives you the most information about your code. However, you must be able to compile the code, and the information is only as up to date as your most recent build.

Tags

CodeWright 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 you can use a hypertext-style lookup of the word at the cursor. This information, too, relies on you updating the database regularly.

Outline Symbols

CodeWright'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 CodeWright.

Objects Window

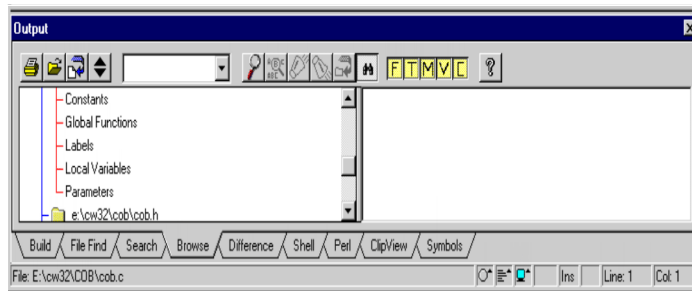
CodeWright's Objects Window is a tab on the Project Window. It displays a hierarchical view of C/C++ and Java symbols that are associated with the name that is typed in the Identifier box at the top of the window. It can be used to view and browse code. Click on a symbol in the Object Window to access the position in the source file at which its corresponding code is located.

Browser Support

The CodeWright Browser supports two kinds of browser databases:

- *Microsoft format .BSC databases* These are the databases generated by Microsoft C/C++ 7.0 and later.
- *Starbase Compiled Tags databases* These are the databases produced by the TAGSWnn utility shipped with CodeWright. The *nn* in the name is replaced by 32. The Tags databases are given the extension .PTG by convention.

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 **Browse** from the **Project** menu or 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 **F1** in the **Browse** window to receive help, press **ESC** at any time to leave the window and return to the current edit window.

Selecting a Database

The first step in using the Browser is to select a database. If you have previously defined a valid .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 may 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, you will note that information about the selected node appears in the Inspect window to the right. This window will list information such as where the item is defined and 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

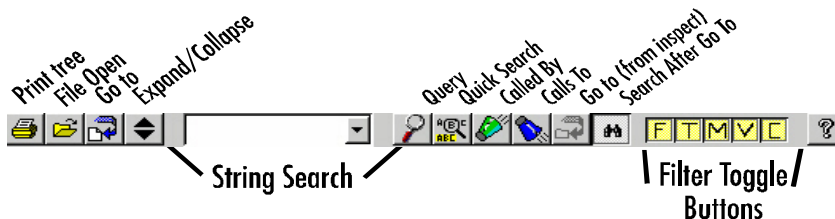
At the beginning of each line of the tree is a bitmapped label that identifies the information in that branch. A key to these labels is given below:

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 will often be labeled <unknown>. These are functions and other objects that are referenced, but whose source was not compiled along with the project. The most common examples of this are the use of functions in the Microsoft Foundation Classes, or other libraries.

Browser Toolbar


Here is how the Browser Toolbar appears:





Refer to this diagram as you read the following descriptions.


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. Simply pressing the  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.

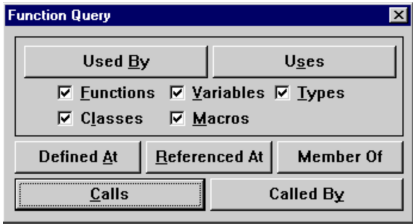
To use the String Search feature, enter a string into the combo box on the browser toolbar. Unlike some other browsers, the string you enter will not be treated in a case sensitive manner, and a trailing wildcard will be assumed. That is to say, if you enter "dump", it will match strings like "Dump" and "dumpFile".

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  key. Depending on the context, one of six query dialogs will appear. An example dialog is shown below:

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.

The six Query Dialogs are as follows:


Button	Purpose
General	General inquiries about the terminals (leaves) on the tree.
Class	OOP class inquiries.
Function	Inquiries about functions.
Friend Class (Fclass)	OOP friend class inquiries.
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 for 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 .

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 CodeWright generated databases; only with .BSC files generated by Visual Studio or with other supported databases generated by more sophisticated, third party browser utilities

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 (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 (i.e.,

).

Below is a list of the buttons and their 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

CodeWright has support for CTags and other Tags-generating programs whose databases conform to the standard Tags format. The file TAGS.C in CodeWright's CWSTART subdirectory (in Full CodeWright installations) contains the functions that support this capability, and also defines the standard Tags file format.

In addition, CodeWright comes with a built-in Tags database generation capability. This program produces both a standard Tags database and a compiled database. As mentioned, the resulting compiled database (.PTG file) may be used with CodeWright'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, CodeWright's tags functions allow you to jump directly to the original location in the original buffer.

CodeWright is known to be compatible with the PCTags program from Moderne Software and GNU Tags. PCTags is available for download from Starbase's FTP site at <ftp://ftp.premia.com/pub/addons/archive/>. GNU Tags is supplied with CodeWright.


Tags Setup

To get CodeWright 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 the chapter on *Projects, Project Spaces, and Workspaces* for information on creating a CodeWright project.
2. Define what files will 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 CodeWright's compiled tags utility, be sure to enter a filename in the **Browser Database** field that employs the .PTG extension rather than .BSC to avoid having the .BSC file overwritten by the browser database generator.
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 **Customize** menu. For more information on the **TagFind** API, refer to CodeWright's online help. For more information on custom keystrokes, refer to the chapter on *Custom 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 (caller/calling trees) are not stored in the Tags database. For more information about the TagsWnn utility provided with CodeWright, see appendix A at the end of this manual.

Outline Symbols: Overview

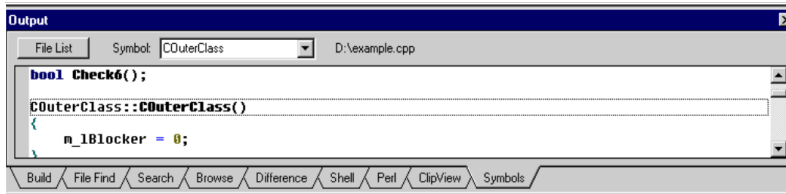
Outline Symbols make up a new type of browser for navigating and investigating code. They are named Outline Symbols because they use two system windows: one called the Outline Window (a tab on the Project Window) and the other called the Symbols Window (a tab on the Output Window). Outline Symbols constantly 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 CodeWright 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. The strings are subsequently displayed 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.

CodeWright'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 in CodeWright's **Edit Symbol File List** dialog. For example, you might want to scan MFC symbols, but don't care to include the MFC source files in each project. You just add them to the **Edit Symbol File List** dialog. To access the **Edit Symbol File List**, press the **File List** button in the Symbols Window. It's as easy as adding files to a project.

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, CodeWright 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 CodeWright 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 (CodeWright 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.

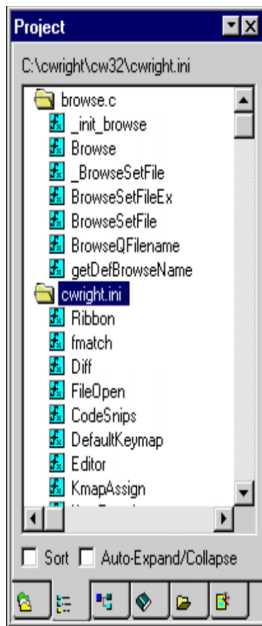
Only if it detects these changes, does it go ahead and start updating the database. If the symbols scan detects a keystroke, it will interrupt its detect scanning and begin again where it left off on the next scan. The symbols scanning and updating can be disabled (see *Symbols Database* below).

Outline Window

CodeWright'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.

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** checkmark 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** checkbox, as described below.

There are two checkboxes at the bottom of this Outline Window tab:

- **Auto-Expand/Collapse** turns off the automatic expansion of the symbols matches. When this checkbox 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 one 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 Parser** 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 checkboxes, 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 unchecking the auto-expand box in the Outline Window, or disabling some of the parser types (see configuration section below).

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 in the **Outline Parsers** dialog. **Symbol Parsers** are regular expression patterns that parse code-elements that match an expression. Patterns can be developed using the CodeWright's Search mechanism. The online help in the **Outline Parsers** dialog provides information on each of the items in this dialog, and provides some tips on creating Regular Expressions. Regular Expressions are also described, under the topic *Regular Expressions* in this chapter. 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.

4. Click **Add**. The **Add Parser Name** dialog comes up.

5. Give the new parser a name and choose a type (i.e. function, prototype, procedure, etc.) from the **Type** drop-down list. The type determines what icon CodeWright uses in the Outline Window.
6. Click **OK**.
7. Back in the **Outline Parsers** dialog, highlight the new parser.
8. 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**.)
9. Choose or enter the characters that should be found by the parser in the **Keyword Characters** section
10. Use the **Keyword Position** options to indicate whether characters to the left or right of the indicated cursor position (^ in the regular expression pattern) should be displayed in the Outline and Symbols Windows.
11. Click **OK**.

The new symbols should appear in the Outline Window when files of the appropriate file type are opened in CodeWright.

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 | Properties | Directories** dialog. By default this database is located in the main CodeWright 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 | Properties | 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, CodeWright automatically updates 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 Windows and the symbols database.

Popup 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 popup 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 CodeWright'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, and types, as they apply to the language being used. Each object-type has its own image, as displayed in the **Filter/Legend** dialog (see the topic *Filter/Legend Dialog* in this chapter).

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 CodeWright's standard popup menu. Those methods are described in the next topics.

Note: The results of either method reflect the **Objects** and **Access** filters marked in the **Filter/Legend** Dialog. If all of the boxes on that dialog are not marked, the caption *Filters Applied* displays after the search criteria at the top of the Object Hierarchy.

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 **Ignore Case** to make the window case insensitive to the characters in the **Identifier** box.

Note: 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, right-click within the Objects Window and select **Ignore CodeSense Library Databases**.

2. Press . A hierarchy will display.

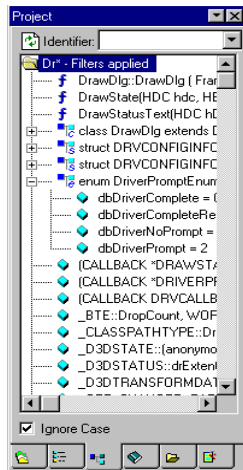
Use CodeWright's Right-Click Popup Menu to Display an Object Hierarchy

To display an object hierarchy by using CodeWright'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.

Objects Window



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 option **Ignore Case** is not marked at the bottom of the window. Mark **Ignore Case** 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 CodeWright.
- A project containing C/C++ and/or Java files must be open in CodeWright.
- 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 the topic *CodeSense* in the chapter *Editing and Printing* for more information.

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:

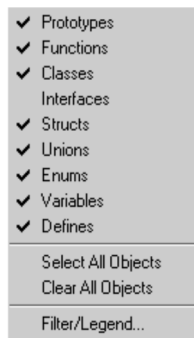
- **Sort Alphabetically** - to sort the objects/symbols alphabetically.
- **Sort by Character Set** - to sort the objects/symbols by the ASCII values of the characters in the name and according to the code page currently in use. This allows the sort to respect locale-specific information (i.e. international 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
- **Show Members in Root** - to show both class detail and function detail if class and function names are the same.
- **Filter/Legend** - to access the **Filter/Legend** popup menu (described next).
- **Refresh Display** - to display the most current list of applicable objects/symbols.
- **CodeSense Editing Options** - to access **Customize|Language|CodeSense**.
- **CodeSense Global Configuration** - to access the **CodeSense Global Configuration** dialog.
- **Ignore CodeSense Library Databases** - to force the **Objects** Window to ignore information in CodeSense Library databases when displaying a hierarchy. This will speed up the process of displaying hierarchies.

Choose the item **Filter/Legend** on the **Objects** popup menu to access the **Filter/Legend** popup menu:

Filter/Legend Popup

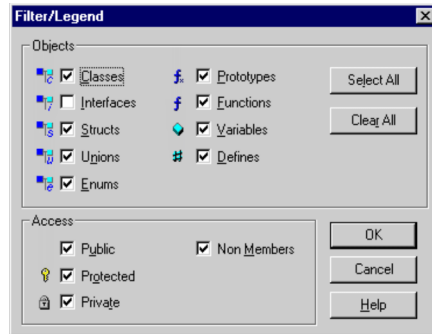


The **Filter/Legend** popup menu lists a set of object filters. Object filters control which objects/symbols will display in the **Objects** Window. Unmark any filter to prevent objects of that type from displaying.

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**.

Filter/Legend



Access filters 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.
- The caption *Filters Applied* displays after the search criteria at the top of the Object Hierarchy if all of the **Objects** and **Access** filters are not selected on the **Filter/Legend** dialog.
- Use the buttons **Select All** or **Clear All** to select or clear selections for all filters.

Symbols vs. Objects vs. Tags

CodeWright offers four browsing features, Symbols, the Object Window, Tags and Browse, all of which have been discussed up to this point. Each browsing method offers its own advantages, some of which are listed next.

Pros and Cons of Tags, Browsers, and Symbols

The following items describe some pros and cons of the different browsing tools available in CodeWright:

- CodeWright's Browser can use Microsoft .BSC files that are produced when code is compiled in Microsoft's development environments.
- CodeWright's compiled Tags databases (.PTG files) provide more limited code-information, but can be built from within CodeWright.
- Both the .BSC and .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.
- Compiled Microsoft Browser .BSC files contain more information about code, such as where symbols are Referenced, Called, etc.
- 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++ 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 and Symbols Windows.
- 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

CodeWright'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. The following paragraphs talk about using CodeWright's 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 .

Setting and Removing Bookmarks

Bookmarks can be set and removed in a number of ways:

- The **Document** menu has a **Mark** option that 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 CodeWright 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 popup 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 **Customize | Environment | Bookmarks** dialog, described next.

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:

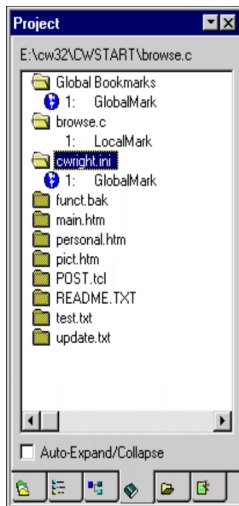
- When bookmarks are visible (i.e. **Show visible bookmarks** options are turned on), a marker appears in the left margin of each line that has a bookmark. (As described in the section *Graphical Bookmark Images*.)
- When prompting is selected, a prompt for an optional bookmark name appears when a bookmark 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.

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 bookmark number,
- The bookmark name, if one was given.

Double-click on the document heading to make it the active edit buffer.

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 datestamp 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** checkbox 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 CodeWright 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. CodeWright 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 will 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.

Using buttons you can perform six different categories of actions:

- **Templates** -- 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 CodeWright templates, see the chapter on *Editing & Printing*.)
- **Macros** -- Macros allow you to 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.
- **Document** -- The document link 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** -- The application link lets you define any arbitrary command line to be executed when the button is pressed.
- **URL** Button Links 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.

12- Checking and Reformatting Files

When you are finished editing a difference, it may be necessary to check and reformat the file. You might also wish to compare and/or merge the finished file previous versions. For these purposes, this chapter covers the following formatting and file-checking tools:

- Differencing
- Merging
- Format Source
- Spell Check

Differencing

CodeWright has a fairly sophisticated differencing feature that can be used directly from within the editor; it is not an external application. CodeWright's 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.

If one's editing requirements include frequent comparisons of files, CodeWright's differencing feature offers an ideal solution.

CodeWright's differencing can be done either side-by-side or interleaved. The following sections describe these two differencing methods.

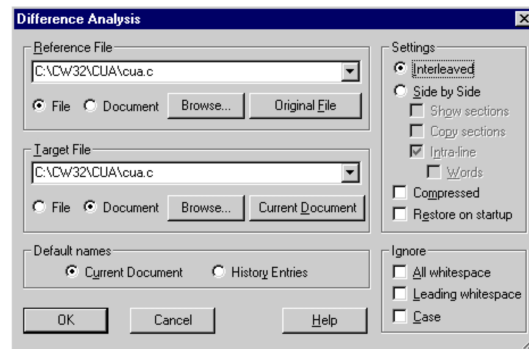
Interleaved Differencing

CodeWright's 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 CodeWright can be initiated from the **File|Difference** dialog. To do an interleaved difference, choose the **Interleaved** option.

Difference Analysis (Interleaved)



Note: 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).

Do the following in preparation for using interleaved differencing:

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 Whitespace:** causes all differences in white space to be ignored.
 - **Ignore- Leading Whitespace:** 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 CodeWright to perform the current difference operation the next time CodeWright 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 CodeWright.
4. 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.
5. 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 CodeWright.
 - 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.
6. Press **OK** to initiate the differencing. The differences will be displayed in a separate, interleaved document in CodeWright. The interleaved document is described next.

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: *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.

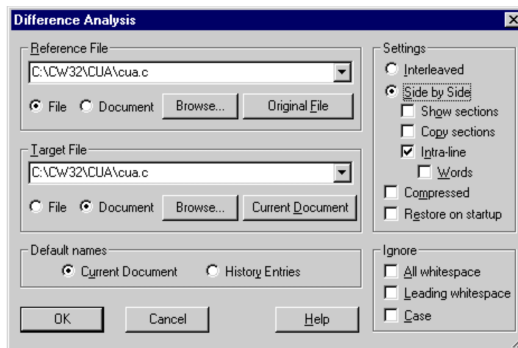
Side-by-Side Differencing

The Side-by-Side difference is shown in a separate tab of CodeWright's 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 CodeWright's Side-by-Side differencing, go to the **File|Difference** dialog and mark the **Side-by-Side** radio button.

Difference Analysis (Side-by-Side)



To perform Side-by-Side differencing, complete the following steps:

1. When **Side-by-Side** is marked, four options will be made available that were not available for the **Interleaved** difference: **Show Sections**, **Copy Sections**, **Intraline**, and **Words**. These options are specifically related to the Side-by-Side differencing. Mark desired options according to the descriptions provided:
 - Mark **Show Sections** to have differing *sections* of the two files being compared completely highlight. If it is not marked, only left margins of the portions that are different are highlighted.
 - Mark **Copy Sections** 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 isn't marked, only one line at a time can be copied.
 - Mark **Intraline** to show the *parts* of the lines that are different, rather than just flagging the whole line as being different.
 - Mark **Words** 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** filenames 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 CodeWright'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.


View Mode Toolbar



There are several buttons on the "view mode" toolbar that are worth describing. The buttons are described below (in order, from left to right).

- The **Question Mark**  accesses **Help** for the 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 from the window in which the cursor sits into a CodeWright buffer. The cursor is positioned at the corresponding location, 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 (i.e. typing) in the Difference Window is not allowed. The Difference Window initially displays in the view mode, as described above. The Change to Edit Mode button  on the Difference Window's toolbar toggles the Difference Window to edit mode, and back.

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 topic *View Mode*, 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 4 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  to jump to the last difference in the two files.
- Press the far-left arrow  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 panes of the Side-by-Side Difference Windows:

-    moves to right window (next).
-    moves to left window (previous).
-    moves to other window (next & wrap).

Key-sequences for the Difference Window can be customized using the function **CWDiffKmapAssign** in the `[KmapAssign]` section of `CWRIGHT.INI`.

Using Difference Utilities

CodeWright 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 CodeWright's **Help|Search For Help On...** dialog. These functions are available for use from CodeWright's **Tools|API Command** key, for binding to custom keystrokes, buttons or menu items, or to use in custom CodeWright DLLs or macros.

Merging

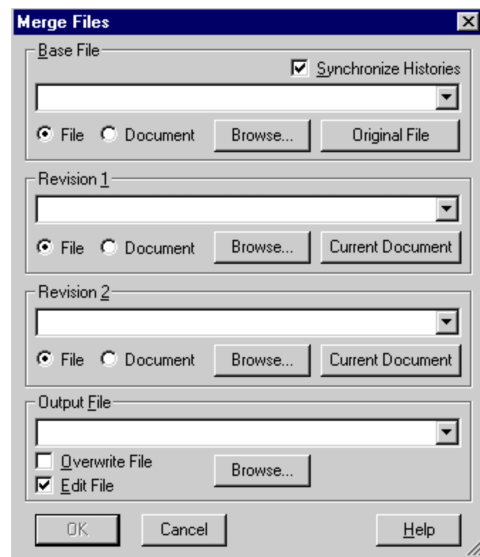
The **Merge Files** dialog allows you to specify three files for comparison, and the name of the file that is to contain the merged changes. By merging files you can determine if there are conflicts between changes made to two or more versions of a file.

Using the Merge Files Dialog

To use the **Merge Files** dialog:

1. Select the **File** menu.
2. Choose **Merge**. The **Merge Files** dialog displays.

Merge Files



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. (Usually this is 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**.

Note: 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 CodeWright 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 CodeWright 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, CodeWright will prompt you for permission to overwrite the file. If you don't give this permission, CodeWright 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 CodeWright 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. Chances are, you will want to look it over in any case. When you do have conflicts, you will need to know how to find them in order to resolve them.

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 lines that need to be compared with those from the **Revision 1** file to see how the conflict should be resolved.

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. (An empty conflict section will represent the revision from which the lines were deleted.)

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: A bug may have been introduced from changes made at some point in the past. Merge allows those changes to be removed without abandoning the changes made since.

Usually, the **Base File** is the oldest of the files used in a Merge comparison, but it need not be. When removing changes, the **Base File** is a middle revision that looks backward at changes to be removed and looks forward at changes to 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.

This may at once appear to be magic, but it is not. 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, CodeWright 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).

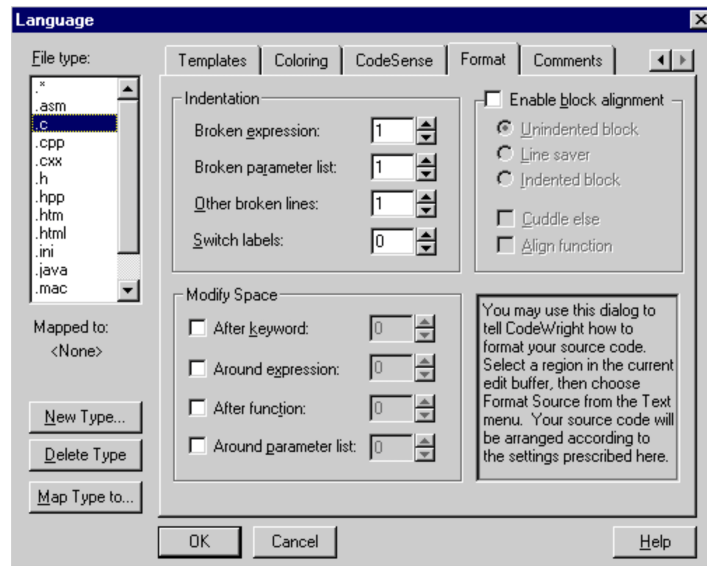
Setting up Your Formatting Criteria

Before you can use the formatting capabilities within CodeWright, you must establish certain criteria for CodeWright 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:

Customize | Language | Format



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 CodeWright 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 CodeWright 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, CodeWright 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 CodeWright 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 CodeWright 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.

Spell Check

CodeWright uses Wintertree Software's Sentry Spelling Checker Engine for its spell-checking operations. The spell-checker is language-specific and supports languages, such as C, C++, Java, Pascal, Basic, and x86 assembly language out-of-the-box. Additional language support can be added with intermediate programming skills in the implementation language of choice and a thorough understanding of CodeWright API extension mechanism. The spell-checker's language variability is limited, by default, to recognizing comment and string components when spell checking is requested to be restricted to those components.

In addition to being language-specific, CodeWright spell checking includes multiple dictionaries, and options for suggesting alternate spellings, replicating capitalization, suggesting split words when accidentally omitting spaces, and suggesting alternate words based on the current word's phonetics. The purpose of this section is to describe CodeWright's spell-checking dialog. For more information on extending the spell checker's capabilities, see CodeWright's online help.

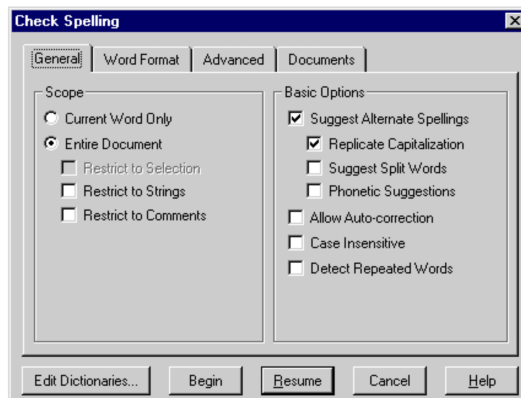
Check Spelling

Click the **Spell Check...** option on the **Tools** pull-down menu to access the **Spell Check** dialog. Extensive online help is available for specific Spell Check options; search for the online topic *Spell Check*. The dialog has four tabs, which are described below:

General Tab of Check Spelling

The **General** tab allows you to set the basic rules for your Spell Check operation.

Tools | Check Spelling | General

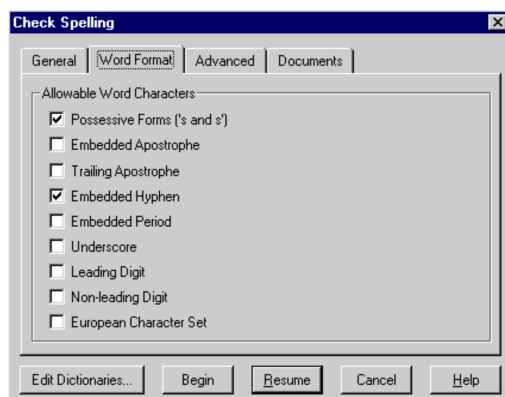


- The **Scope** section of the **General** tab offers the option of checking the **Current Word** only, or the **Whole Document**. If the choice is to check the **Whole Document**, there are additional options for restricting the scope to a **Selection** only, **Strings** only, or **Comments** only.
- The **Suggest Alternate Spellings** option determines whether similarly spelled words should be listed if the word being checked is not found in the dictionary. When this option is enabled, the **Replicate Capitalization**, **Suggest Split Words**, and **Phonetic Suggestions** options become available.
- The **Allow Auto-correction** option enables/disables the auto-correction feature. (If turned off, an auto-correct entry in the dictionary will be treated as if it is a conditional replacement.)
- The **Case Insensitive** option retains the case of the word being replaced.
- The **Detect Repeated Words** option allows for detecting accidentally repeated words such as 'the the' and suggesting 'the' as a replacement. The detection will occur only if there are no characters other than space and tab between the words.

Word Format Tab of Check Spelling

The **Word Format** tab lets you control what characters are accepted when words are being retrieved for checking. The options chosen here, along with options chosen in the **Scope** section of the **General** tab, govern how the spell checker retrieves a word for checking. They do not affect the decision the spell checker makes to ignore a word (i.e. to consider a word correctly spelled). Characters that qualify for inclusion are collected based on these options. The character collection continues until a non-qualifying character is encountered.

Tools | Check Spelling | Word Format



The options are:

- **Possessive Forms ('s and s')** - When enabled, the indicated sequences ('s and s') are included as parts of words to be checked, but only if they occur at the end of the word and are preceded by a letter. Use this option if you only want apostrophes to be considered parts of words when they are immediately post- or preceded by 's'.
- **Embedded Apostrophe** - When enabled, apostrophes are included as parts of words to be checked, but only when the apostrophes occur between two letters.
- **Trailing Apostrophe** - When enabled, apostrophes are included as parts of words to be checked, but only when the apostrophes occur at the end of words. This option differs from **Possessive Forms** in that it considers apostrophes viable word-characters even if the character that precedes them is NOT an 's'.
- **Embedded Hyphen** - When enabled, hyphens are included as parts of words to be checked, but only when the hyphens occur between two letters.
- **Embedded Period** - When enabled, embedded periods are included as parts of words to be checked, but only when the periods occur between two letters.
- **Underscore** - When enabled, underscores are included as parts of the word to be checked. They are therefore included in the set of characters that the spell checker considers when discerning what words need to be checked.
- **Leading Digit** - When enabled, words that begin with a digit are included as parts of words to be checked.
- **Non-leading Digit** - When enabled, words with embedded digits (i.e. digits that appear anywhere other than in the word's first position) are included as parts of words to be checked.
- **European Character Set** - When enabled, alphabetic characters above 0x7f (European character sets) are included as letters for the spell checker to check, additionally 0x92 may serve as an apostrophe.

There is also an API that allows the specification of additional characters that should be considered alphabetic and therefore allowed in words. The API is

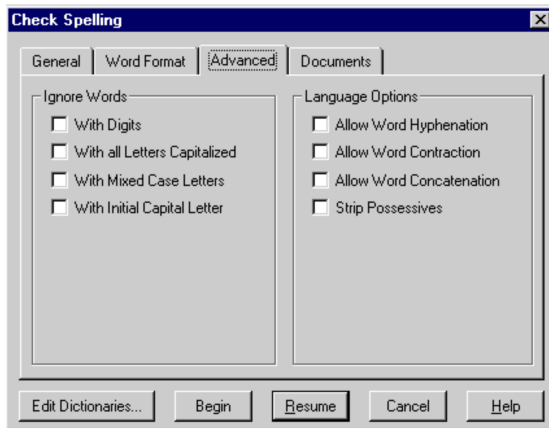
```
void SpellWordChars(LPSTR charSet);
```

For more information on CodeWright APIs, consult CodeWright's online help.

Advanced Tab of Check Spelling

The **Advanced** tab has options that specify that words containing certain character sequences should be considered correctly spelled. Note that some of the options in the **Advanced** dialog will only work if corresponding options on the **Word Format** tab are also enabled. The options on the **Word Format** tab tell the spell checker to include certain characters when deciding what characters are parts of words. If the spell checker does not know that those characters are words, it won't use the options on the **Advanced** tab to spell check them. For example, marking **With Digits** tells the spell checker to check the spelling of words containing digits. However, it will only work if digits are first known to be parts of words. Therefore, when **With Digits** is marked, either or both **Leading Digit** and/or **Non-leading Digit** must also be marked on the **Word Format** tab.

Tools | Check Spelling | Advanced



The options in the **Ignore Words** section are:

- words **With Digits**
- words **With All Letters Capitalized**
- words with unusual capitalization, or **With Mixed Case Letters** (e.g. BlueSky)
- words with **Initial Capital Letters** (first letter capitalized)

The **Language Options** section controls how the spell-checker processes words when searching for them in the dictionaries. The options are:

- **Allow Hyphenation** - When enabled, the spell checker will check the spelling of all parts of words containing hyphens. If all the parts are correct, the whole word is considered correctly spelled, and is therefore ignored by the spell checker. This option will only work if **Embedded Hyphen** is also marked on the **Word Format** tab.

- **Allow Contraction** - When enabled, the spell checker will spell all parts of words containing apostrophes. If all the parts are correct, the whole word is considered correctly spelled, and is therefore ignored by the spell checker. This option will only work if **Embedded Apostrophe** is also marked on the **Word Format** tab.
- **Allow Concatenation** - When enabled, the spell checker will look for component words within a word. For example, braveheart would be considered correctly spelled because it consists of two correctly spelled words. This is most useful with languages such as German that allow ad hoc concatenation of words to form new words.
- **Strip Possessives** - When enabled, the spell checker will remove 's and s' from the ends of words before checking them. This is most useful with English and should be used in conjunction with the necessary word format options.

Possessives are stripped from words with forms similar to the following:

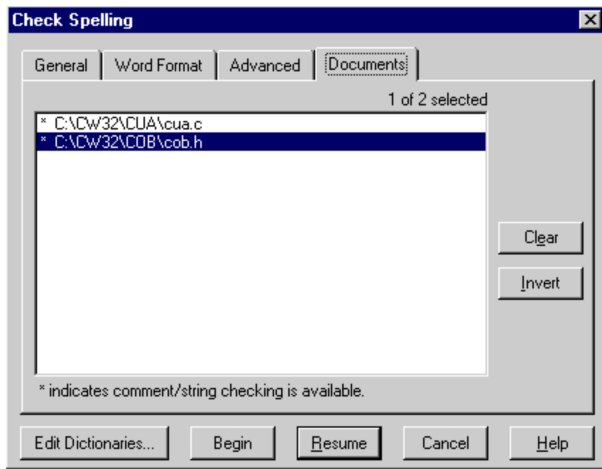
Example: Bordeaux' = Bordeaux
 Lions' = Lion
 Children's = Children

Strip Possessives will only work if the following option or combination of options is enabled on the **Word Format** tab:

- ✓ **Possessive Forms**, or
- ✓ Both **Embedded Apostrophe** and **Trailing Apostrophe**, or
- ✓ **Possessive Forms**, **Embedded Apostrophe**, and **Trailing Apostrophe**.

Documents Tab of Check Spelling

The **Documents** tab of the **Check Spelling** dialog displays a list of the files currently open in CodeWright. Files can be selected from the list for spell checking. All documents can be selected by clicking **Invert**. Selections can be cleared using the **Clear** button. An asterisk (*) preceding the file name indicates that language-dependent comment/string checking is available.



Dictionaries Dialog

Clicking the **Edit Dictionaries** button in the **Check Spelling** Dialog accesses the **Dictionaries** Dialog. The **Dictionaries** dialog is used for adding and editing dictionaries that will be used by the spell-checker. The **Dictionaries** dialog imparts the ability to do the following:

- Designate which dictionary to use for certain documents.
- Transfer words between the currently selected dictionary and a text file using the **Import** and **Export** buttons.
- Create new spell-checker dictionaries on the fly in the **New Dictionary** dialog (accessed by clicking the **New File** button in the **Dictionaries** dialog).

Dictionaries

The Sentry Spelling Checker engine uses both text and compressed dictionaries. Some dictionaries are contained in the *Spellchk* subdirectory of the CodeWright home directory. The text dictionaries are faster to access but they use disk space less efficiently for the amount of words they hold. For the standard language dictionaries, the text dictionary contains hundreds of short and commonly used words. The compressed dictionaries contain 100,000 or more other words. Text dictionary entries should be in alphabetical order and are editable using any text editor.

These are the dictionaries that are currently available for CodeWright's spell checker:

- ACCENT.TLX is a dictionary for English words containing accented characters.
- CORRECT.TLX is an optional dictionary containing several hundred common misspellings and their auto-correct replacements.
- HTML.TLX is an optional dictionary that gives HTML keywords.
- SSCEAM.TLX and SSCEAM2.CLX are text and compressed dictionaries, respectively, for American English.
- SSCEBR.TLX and SSCEBR2.CLX are text and compressed dictionaries, respectively, for British English.
- SSCEFI.TLX and SSCEFI2.CLX are text and compressed dictionaries, respectively, for the Finnish language.
- SSCEFR.TLX and SSCEFR2.CLX are text and compressed dictionaries, respectively, for the French language.
- SSCEGE.TLX and SSCEGE2.CLX are text and compressed dictionaries, respectively, for the German language.
- SSCEIT.TLX and SSCEIT2.CLX are text and compressed dictionaries, respectively, for the Italian language.
- SSCESP.TLX and SSCESP2.CLX are text and compressed dictionaries, respectively, for the Spanish language.
- SSCESW.TLX and SSCESW2.CLX are text and compressed dictionaries, respectively, for the Swedish language.
- USERDIC.TLX is a sample user dictionary.

For more information on CodeWright dictionaries and their use, refer to subtopics under the index heading *Spell Check* in the online help.

13- Custom Interface

This chapter discusses:

- Dockable toolbars and windows.
- Customizing and manipulating CodeWright's toolbars, buttons and menus.
- Choosing and using keymap command sets.
- Using and customizing mouse commands.
- Reassigning keystrokes and mouse actions.
- Recording keystroke macros.

Dockable Toolbars and Windows

CodeWright has a number of 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 **Customize | Toolbars** dialog is where toolbars can be turned on, created or modified, or otherwise manipulated.

Toolbars

Several default toolbars are available in CodeWright'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 are:

- **Hide when application is inactive:** The toolbar is only visible when CodeWright 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, 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 next.

- **Standard Toolbar:** The **Standard** Toolbar is turned on in CodeWright by default and docked at the top of the CodeWright screen, under the main menu. A description of the **Standard** Toolbar is provided in the *Overview* chapter towards the beginning of the manual, under the section entitled *A First Look*.
- **Build Toolbar:** The **Build** Toolbar has buttons that can be used to run some of the commands, like **Compile** and **Build**, that have been set up in **Project | Properties | Tools**.
- **Edit Toolbar:** The **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:** The **Tools** Toolbar has several buttons that perform various tasks, like **File Find**, **Differencing**, **Merging**, etc.
- **VCS Toolbar:** The **VCS** Toolbar has buttons for performing various operations with your defined version control program.
- **HTML Toolbar:** The **HTML** Toolbar is only available when **HTML WYSIWYG Editor/Viewer** is marked in **Customize | Libraries**. It has buttons that perform operations specifically designed for HTML files.
- **AppBasic Toolbar:** The **AppBasic** Toolbar has buttons that perform operations that specifically apply to CodeWright's AppBasic macro language, described in the chapter *Extend CodeWright*.
- **Project Toolbar/Window:** The **Project** Toolbar is actually a window that is turned on and docked, by default, on the left-hand side of the CodeWright screen. It has six tabs, **File View**, **Outline**, **Objects**, **Bookmarks**, **File Open**, and **CodeFolio**, which are described in various sections of this manual.
- **Output Toolbar/Window:** The **Output** Toolbar is actually a window that is turned on by default. It initially has seven tabs and is docked on the bottom edge of the CodeWright screen. Three more tabs can optionally be added by loading Add-Ons for the **AppBasic** and **Perl** macro languages, and for the **Clipboard/Scrap Viewer**. Add-Ons are loaded using the **Customize | Libraries** dialog. The tabs of the Output Window are described in various chapters in this manual.
- **MSDevSync:** The **MSDevSync** Toolbar has buttons that aid the MSVC++ 6.0 synchronization program. The buttons will execute the MSDev compile, build, and toggle breakpoint commands, and synchronize the files open in CodeWright with the MSDev IDE. More information on the toolbar can be found in CodeWright's online help. CodeWright's Sync technology is described in the chapter on *Synchronization*.

- **TICCSync:** The **TICCSync** Toolbar has buttons that aid the Texas Instruments Code Composer Studio synchronization program. The buttons will execute the Code Composer compile and build commands, and they will synchronize the files open in CodeWright with the Code Composer IDE. More information on toolbar can be found in CodeWright's online help. CodeWright's Sync technology is described in the chapter on *Synchronization*.

Auto-hide Toolbars

CodeWright 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. CodeWright'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 CodeWright 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**. Mark **Display Docked Toolbar Titles** to have the names of auto-hidden toolbars display in the hidden toolbar's border.

What Does Dockable Mean?

A dockable window or toolbar can either be attached to one of the edges of CodeWright'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 may be placed anywhere on the screen and may be resized to almost any shape, but can partially obscure other objects.

Toolbar Docking Precedence

The **General** tab of the **Toolbars Customization** dialog (**Customize | Toolbars**) displays a conceptual image of the four CodeWright window-edges that allow toolbar-docking. The overlapping edges in the image indicate toolbar-docking precedence. Toolbars that are docked on edges with precedence will 'push over', or overlap toolbars that are docked on edges without precedence. Toolbar docking precedence allows toolbars to take full advantage of the horizontal or vertical expanse of a preferred edge.

Enabling and Disabling Toolbars

Once any dockable toolbars or windows are enabled (visible), a 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 dockable windows will pop up. The items preceded by checkmarks are the ones that are currently visible.
2. To change the status of any of toolbar or dockable window, select it from the list.

If no toolbars or dockable windows are currently visible, do the following:

1. Select the **Customize** menu.
2. Choose **Toolbars**.
3. Select the desired toolbar from the list.
4. Check the **Visible** checkbox and press **OK**. 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 will become free-floating.
- If it was free-floating, it will become 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:

- Drag a free-floating window 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) to dock it. It will automatically dock when placed over the edge of the client area.
- Drag a free-floating window by its title bar to keep it from docking.

When you drag a free-floating toolbar or window to the edge of CodeWright'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:

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 CodeWright 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  key.
 3. Let go of the mouse button.
 4. Let go of the  key. The toolbar will be undocked.
 5. You can then resize and reposition the free-floating toolbar or window at will.

Customizing Toolbars and Buttons

CodeWright has a number of standard toolbars that can be modified through a simple drag and drop interface. You can also:

- 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. Click on the **Toolbars** tab.
3. Press the **New** button. The **New Toolbar** dialog displays.
4. Enter the name of your new toolbar and press **OK**.

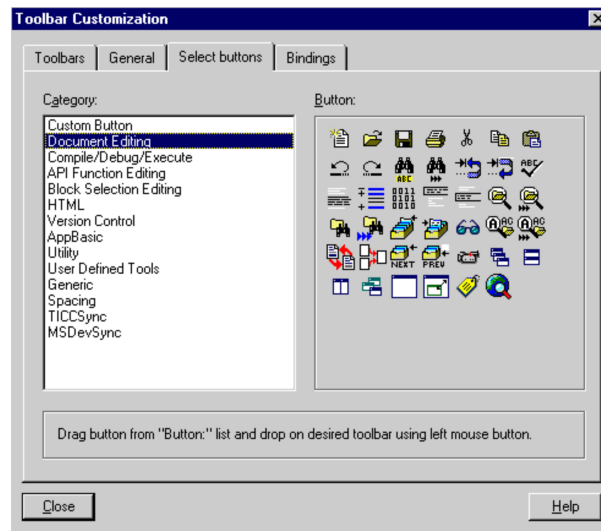
CodeWright then creates a new toolbar for you. At this point the toolbar is empty. You will need to move it, dock it, etc. You will also need to add buttons to it. The latter is covered in the next topic, *Adding and Changing Toolbar Buttons*. The toolbar information is stored in the CWRIGHT.INI file.

Adding and Changing Toolbar Buttons

To add or change a toolbar button:

1. Go to **Customize | Toolbars**, or right click on any visible toolbar button, to access the **Toolbar Customization** dialog.
2. Click on the **Toolbars** tab.
3. Select the toolbar you wish to modify; the toolbar will be displayed in a separate window.
4. Click on the **Select buttons** tab. On the **Select buttons** tab of the **Toolbar Customization** dialog, CodeWright enters a special mode where the toolbars are temporarily inoperative.

Toolbar Customization: Select Buttons



5. From this dialog you can:
 - 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.

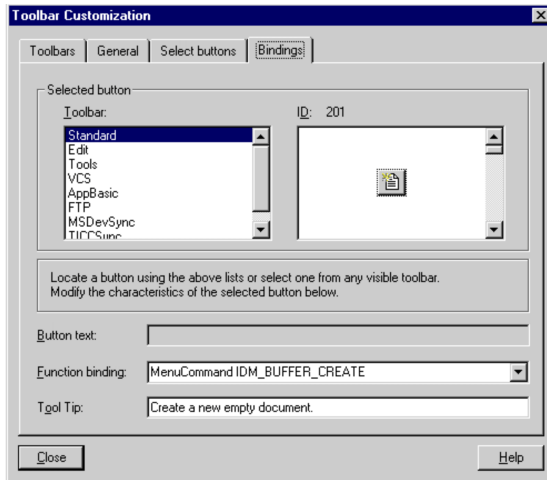
There are several categories of buttons in the dialog, listed in the **Category** window. The category **User-Defined Tools** includes buttons with no pre-defined function bindings, to which you can quickly bind your function.

Binding a Function to a Button

To bind a function to a toolbar button:

1. Go to **Customize| Toolbars**, or right click on any visible toolbar button, to access the **Toolbar Customization** dialog.
2. Click on the **Bindings** tab.

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 CodeWright and follow the rules of **LibFunctionExec**.
 - The **Tool Tip** field contains the message that pops up when the mouse cursor pauses over the button.


Notes:


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 CodeWright directory. CodeWright takes care of updating this file, much like an .INI file, as buttons are customized.

All bitmaps that are available in the dialog for the buttons come from the CWDLL32.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 CodeWright'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.

- Press the up arrow to reuse the most recently entered response.
- Press the down arrow to reuse the oldest response.
- Press the up arrow repeatedly to view earlier responses.
- Press the down arrow repeatedly to view more recent responses.
- Press  to accept 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. press  to accept the edited response at anytime. 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 isn't in the history.

Editing Combo box History Lists

CodeWright has a **History** tab in the **Customize|Environment** dialog that 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 CodeWright sessions by default. The information is saved in the State file. For more information about CodeWright State files, refer to the *Configuration Files & Command Line Parameters* chapter of this manual.

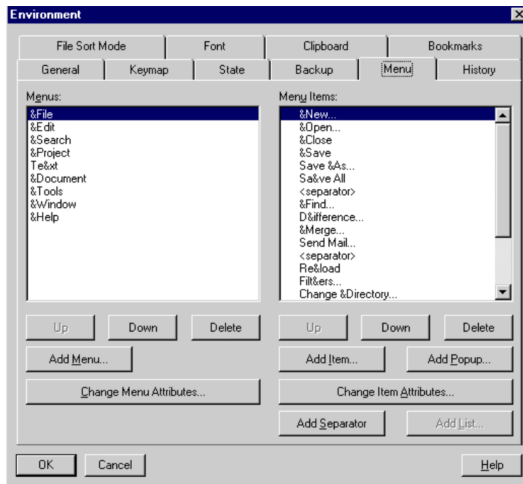
Customizing Menus

This section describes how to edit standard menus, submenus, menu items and popup menus.

Menu Editor

You can change, delete and add menus and menu items through the Menu Editor on the **Customize | Environment | Menu** dialog.

Menu Editor



The dialog lists **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:** These buttons 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:** The **Add Menu** button/dialog lets you add a menu to CodeWright'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:** The **Change Menu Attributes** button/dialog is like 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:** The **Delete** button 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 CodeWright's configuration file, CWRIGHT.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 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:** These buttons 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:** The **Add Item** button/dialog lets you add an item to one of CodeWright'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:** This button 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 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:** The **Delete** button 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:** The **Add Separator** button 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.

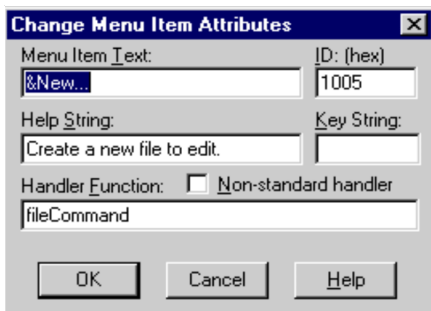
Whenever you are through operating on a submenu, you may move it back to the list on the right by double clicking on the name again. The minus sign will change back to a plus when the submenu arrives in the list on the right. This system allows you to have any level of nesting of submenus that you like. 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 on the **Menu** tab.
3. Click on a menu, and corresponding menu item to change.
4. Press the **Change Item Attributes** button.

Change Menu Item Attributes



This dialog shows you information about the selected menu item such as its **ID** in Hex, the function bound to it, etc. 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:

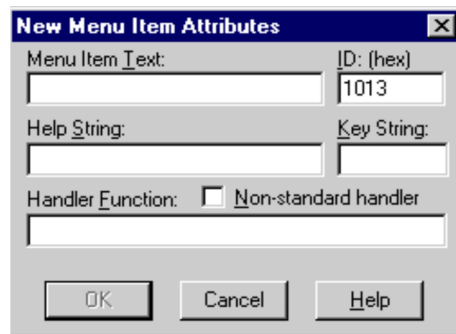
- Bind the menu item to a function that begins with an uppercase letter. Any of the CodeWright API functions should work, or your own functions (or macros), provided they have met the criteria to be recognized by CodeWright (See the topic *Exporting Functions* in the chapter *Extend CodeWright*).
- Check the small checkbox labeled **Non-standard handler** when binding functions that do not have a menu handler. You must check this box, or CodeWright won't handle your customization, and your menu items won't work.

Adding a Menu Item

To add a new menu item, complete the following steps:

1. Select the **Customize|Environment** dialog.
2. Click on the **Menu** tab.
3. Select the menu to which the item should be added.
4. Press the **Add Item** button. The **New Menu Item Attributes** dialog displays. This dialog is the same as the **Change Item Attributes** dialog.

New Menu Item Attributes



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.

Note: 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. Check the box labeled **Non-standard handler**.
10. Press **OK**. CodeWright assigns the menu ID in hex to the new item.

Note: 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.

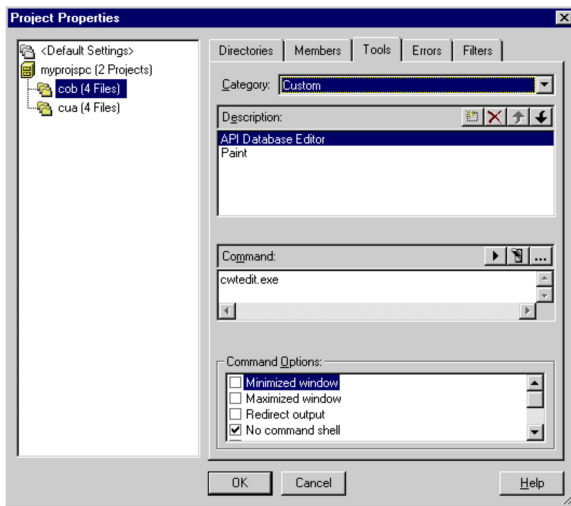
Customizing External Operations within CodeWright

As described in the chapter on *Projects, Project Spaces, and Workspaces*, CodeWright 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|Properties** dialog.
2. Click on the **Tools** tab.
3. Choose the **Custom** option from the drop down under the **Category** combo box. The dialog is displayed next.

Project Properties Tools - Custom Category



4. In the **Command:** edit 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 CodeWright will shell to DOS in the directory specified on **Project|Properties|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 CodeWright 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 CodeWright's **Popup Menu Editor**. The following paragraphs describe CodeWright's **Popup Menu Editor** and provide steps for creating and editing popup menus.

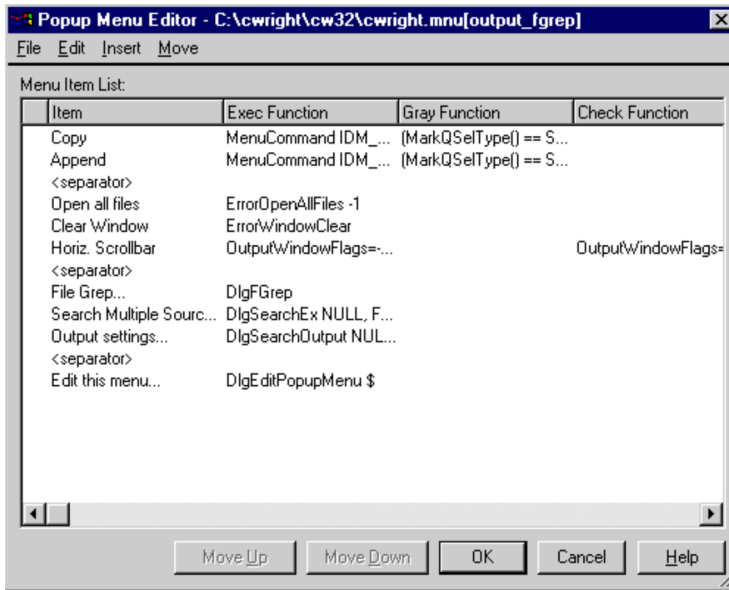
Clicking the **Edit this menu...** item on most CodeWright popup menus accesses the **Popup Menu Editor**. The function **DlgEditPopupMenu**, used from CodeWright's **API Command** dialog, a keystroke, a button, or a menu item, will also access the editor.

Editing or Creating a Popup Menu

Do the following to edit or create a popup menu in CodeWright:

1. Either access **Edit this menu** from any CodeWright popup menu, or type:
`DlgEditPopupMenu`
from the **Tools|API Command** dialog.
2. The **Popup Menu Editor** should appear with CodeWright's default menu definition file CWRIGHT.MNU open.

Popup Menu Editor



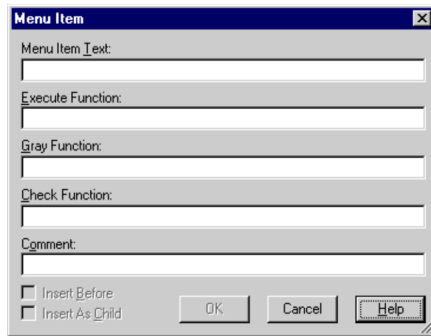
3. In the **Popup Menu Editor** click **File|New** to create a new menu, or click **File|Open Menu File** to open an existing menu. If a new menu is being created, a new, empty menu will be opened in the editor. If an existing menu is opened, the existing menu will appear in the editor.

Note: When opening an existing popup menu for editing, the **Open Menu or Menu File** dialog appears. The name of the menu description file 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.

Do the following to add or modify a menu item in the popup menu:

1. In the **Popup Menu Editor**, 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.

Menu Item



2. Fill in the necessary fields in the **Menu Item** dialog. The uses for the fields are as follows:

- **Menu Item Text:** This is 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. Press the key for that character while the menu is visible to access the item.
- **Execute Function:** This is the function that will be used for normal menu items. The functions are called when the menu item is clicked. Most exported CodeWright functions can be used.

Example: The following item can be added to any popup menu for opening files:

```
&Open FileBufEditFile
```

&Open File is the menu item string, and **BufEditFile** is the execute function.

- **Gray Function:** Indicates availability of the menu item's function. When the menu item is gray, the function is not available.

Example: The **Terminate** item on the **Perl** popup menu is gray when a Perl macro is not running. When enabled, it is used to forcibly terminate a running Perl macro. The following function is entered in the **Gray Functions** field:


```
!_PerlCtrl -1
```

- ✓ If the function returns 0 (false), the **Terminate** menu item will be gray.
- ✓ If the function returns 1 (true), the **Terminate** menu item is enabled.
- ✓ The exclamation point (!) before the **_PerlCtrl-1** function indicates that the complement of the function's return value should be used when the function is called.

- **Check Function:** These correspond to enabled menu items. A checkmark to the left of the menu item indicates that the item is enabled.

Example: The **Debug Mode** item on the **Perl** popup menu toggles the Perl interpreter's Debug mode. The item is not checked if Debug Mode is turned off. The following function is entered in the **Check Functions** field:

```
PerlFlags 2 -1
```

- ✓ If the function returns 0 (false), the **Debug Mode** menu item will be unchecked.
- ✓ If the function returns > 0 (true), the **Debug Mode** menu item will be 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. If no menu file is specified, CodeWright's default menu file CWRIGHT.MNU is used. CWRIGHT.MNU is stored in the CodeWright installation directory. It contains sections as a normal .INI file does, except that the contents are menu descriptors. 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. As with .INI files, lines beginning with a semicolon and blank lines are ignored.

Before creating or modifying popup menus in CodeWright, 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 the **Popup Menu Editor's Menu Item** dialog:

- 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 CWRIGHT.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's looking for the end of a section.

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 the **Popup Menu Editor's Insert** menu 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 Popup Menu Functions

There are two functions that support the operation of popup menus. They are as follows:

- **DlgMenuPopup** -- Use this function to assign a popup menu to a key or mouse click. The function is formatted as:

```
DlgMenuPopup <menu>
```

- **DlgMenuExec** -- Use this function to execute other sections within the CWRIGHT.MNU file. The function is formatted as:


```
DlgMenuExec <section>
```

Where:

<menu> - If the name starts with '[', it is assumed to be a section in the .MNU file, otherwise it is the name of the file to use (no sections in file).

<section> - A normal CodeWright section to execute out of the .MNU file. Use **ConfigFileRead()** to read out of other files.

Refer to the following example:

Example: The [Utilities] section of CWRIGHT.MNU can be bound to -right-mouse click through the following key binding command:

```
KmapAssign='<Shift-Mouse_right_click>'
'DlgMenuPopup [utilities]'
```

(See the topic in this chapter *Customizing with Keybindings* for keybinding information.)

Using Keymaps

The CodeWright editor is several editors in one. In its "standard" mode, the CodeWright 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.

CodeWright 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, CodeWright 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, from Underware)
- 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 using the **Customize|Keyboard** dialog (see *Binding Keystrokes to Functions or Macros*).

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. (It's hard to associate anything with , you just have to memorize it.) 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 . To *find*, press .

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  key combinations. CUA rules reserve most of these for standard commands, such as accessing menus. For instance,   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  key to access the menu. To access the **File** menu, for example, you press and release the  key and then press . Similarly, to access other menus, you press and release  and then press the underlined letter in the menu's name.

Mouse Commands

You can probably guess most of the things that you can do with a mouse in CodeWright: click on menus, select text and so on. There may be a few things that you wouldn't guess you could do with a mouse, and other things that you might suspect you could do but do not know how to do. It is those things that are covered in this section.

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, CodeWright 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, e.g. full screen mode. CodeWright provides some methods for gaining control over scrolling speed in these cases. Control is gained via the , , and  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 8 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 may 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 a mouse or other selection open or closed.

Examples:

In the BRIEF-compatible keymap, toggle the selection open or closed using the  command.

In the CUA keymap, toggle the selection open or closed using the  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.

Note: 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 popup menus that can be accessed just by clicking with the right mouse button on hotspots on the status line. These actions are listed below:




Action	Description
Insert/Overtyping Toggle	Toggle between insert and overtype mode by right clicking on the Ins or Ovr 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 Go to Line dialog by right clicking in the Line number box.
Next Message	Have CodeWright process the next Build error message by right-clicking in the message box at the left of the status line.
	Bring up a customization popup menu with options for accessing various dialogs used for customizing CodeWright.
	Bring up a Word Wrap Options menu to quickly toggle wrap mode and customize wrap options.
	Bring up a CodeSense customization menu to quickly turn on and customize CodeWright's CodeSense and Outline Symbols features, as described in the chapters on <i>Editing & Printing</i> and <i>Search and Replace and Navigational Tools</i> .

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:
 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 text cursor follows the mouse cursor, indicating where the destination will be. Place the caret at the intended destination and release the mouse button to move the selected text.
- To move text between windows, complete the steps above but press  as you drag. Be sure to click and hold the mouse button before pressing the  key. Then drag and release the text.
- To perform a copy operation, complete the steps above, but press the  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, deselect the **Drag and drop text with mouse** option on the **Customize|Environment|General** dialog.

Mouse Copy and Move

In addition to Text Drag and Drop, CodeWright 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  . The copy command is   .

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 so long as you are using closed selections for mouse operations, it won't 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 onto your current buffer, so that you can view two different sections of the buffer at the same time.

To create a window:

1. Point to any part of CodeWright's Client Area that is not currently occupied.
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. This use of the left mouse button can be turned off in **Customize|Environment|General**. Just deselect the option **Create Windows Using Mouse**.

Drag-and-Drop File Loading

The Drag and Drop feature allows you to load files into CodeWright from Windows Explorer. 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.

Drag and drop file loading can also be used to load or create projects and project spaces. The feature works when files with .PSP, .PJT, .DSW, or .DSP filetype extensions, or any supported makefile, are dropped from Windows Explorer onto CodeWright. **Auto-detect file type** must be marked in **Customize|Environment|General**.

If you already have files loaded in CodeWright, the files you drag and drop onto CodeWright will be added to those already being edited:

- A window is created for each of the files dropped onto CodeWright 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.
- Files with .PSP, .PJT, .DSW, or .DSP filetype extensions, or any supported makefile will load or create CodeWright project spaces or projects. **Drag and drop project files** must be marked in **Customize|Environment|General**.

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 the chapter on *Search and Replace and Navigational Tools* 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

Now that you learned about CodeWright's keymaps and mouse commands, you are ready to make assignments specific to your keymap. Your main tool for doing this will be CodeWright's **Assign Keys** dialog. Keys can be modified using CodeWright 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 CodeWright's status bar when the keystroke is pressed. Refer to the chapter on *Extend CodeWright* for more information on custom DLLs or Add-Ons.

Customizing with Keybindings

As mentioned in the section on *Using Keymaps*, CodeWright 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. We have additional keymaps available in our Technical Support Add-Ons collection or you can create your own.

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 found one. 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 how to make keystroke macros in CodeWright. Note that keystroke macros are limited to actual keystrokes (mouse clicks won't be accepted) and keystrokes entered into Windows dialog boxes cannot be recorded.

Recording a Keystroke Macro

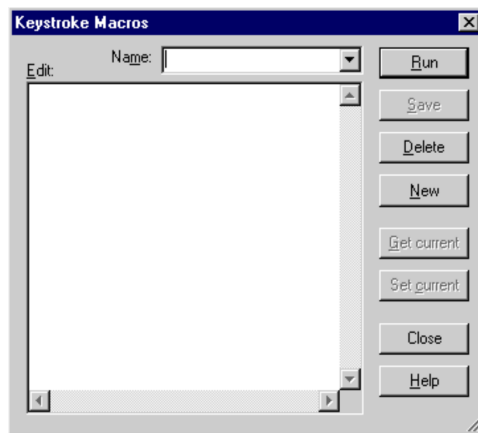
You can record a keystroke macro in any of the following ways:

- Press **F7** (or **CTRL|X** 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.

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**.

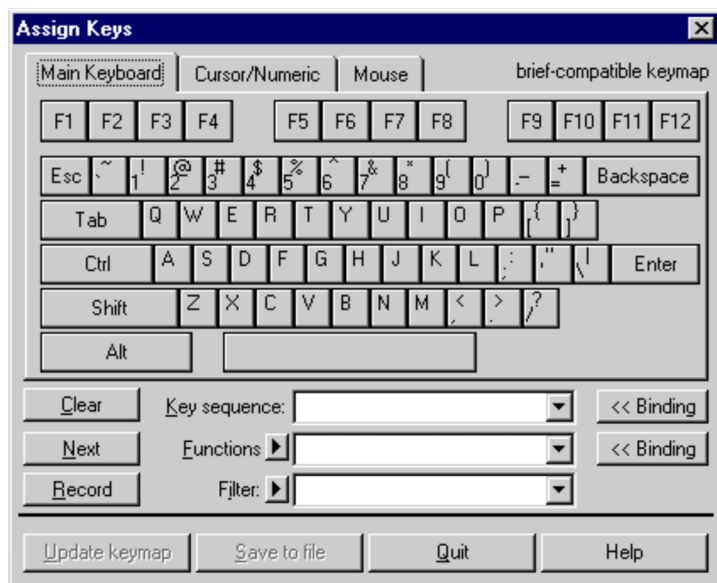
Once the keystroke macro has been saved with a name, it can be used from CodeWright's **Tools|API Command** dialog, or bound to a keystroke, button, or menu-item simply by typing the name of the macro in the appropriate place. Named keystroke macros can also be used in CodeWright's **AppBasic** macros, **Perl** macros, and **API** macros. For more information on macros, see the chapter on *Extend CodeWright*.


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.

Customize|Keyboard



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 mouse-click the key representations in the proper sequence.
3. Verify that the key sequence isn't already assigned by pressing **Binding** next to the **Key Sequence** box.
4. Locate the macro (or function) to bind from the drop-down list in the **Functions** box. To narrow your search, you can:
 - Choose or enter a filter in the **Filter** box. For example, to see only functions that begin with CUA_ in the drop-down, type *CUA_*.
 - To have only the functions in a selected category display in the drop-down list, press the right arrow  next to the **Functions** box, and choose either **Functions** or one of the macro categories.
5. Verify that the function or macro doesn't already have a key binding by pressing the **Binding** button to the right of the **Functions** box.
6. 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.

14- File Loading, Backup and FTP

This chapter discusses options and strategies for file loading and validation, and for file backup and auto-save. It includes a brief run-down of CodeWright's FTP utility, which can be used for transferring files to and from remote systems from within the editor.

File Loading, Reloading and Validation

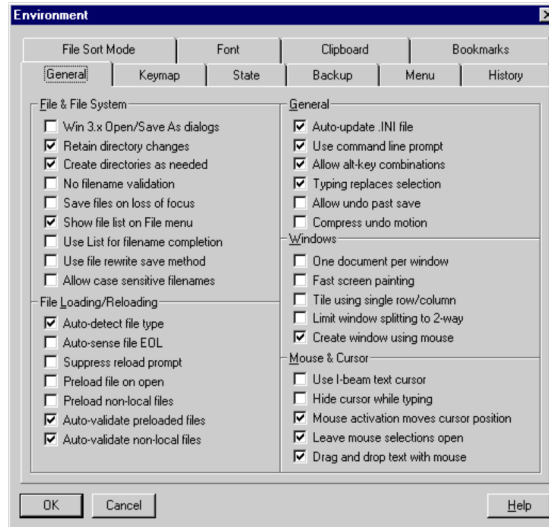
CodeWright has options for preloading and auto-validating local and non-local files. Read on to find out what these terms mean, and how to use the features that enable them.

- A file is preloaded in CodeWright 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 CodeWright 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. Access the **Customize|Environment|General** dialog.

Customize | Environment | General



2. The following options in the **File Loading/Reloading** group can be enabled as precautionary measures to maintain the integrity of files loaded in CodeWright:

- **Auto-sense file type EOL:** When this box is checked, CodeWright 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  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.

Note: It is advisable to use the **Auto-sense file type EOL** option when editing UNIX files in CodeWright.

- **Suppress reload prompt:** If another process alters a file that you have loaded into CodeWright, 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, CodeWright 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, CodeWright 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.
3. The following options in the **File and File System** group can be enabled for quick access to files being loaded in CodeWright and as precautionary measures to maintain the integrity of files being edited:
- **Show file list on File menu:** When this box is checked, CodeWright 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 CodeWright 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, CodeWright 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, CodeWright 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 the Chapter on *Large Files* for more information.

File Backups and Auto-save

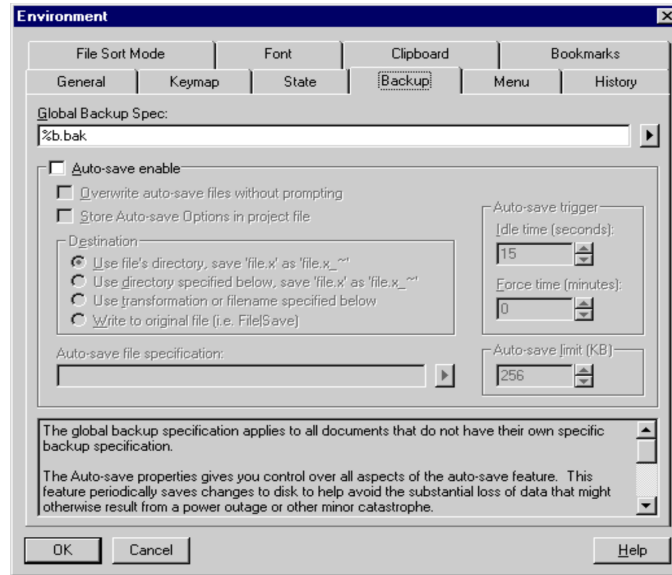
This section discusses how to set up file backup and auto-save options in CodeWright. The options can be set globally, for all files being edited, per file type (e.g. .CPP, .C, .TXT), or per individual file.

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|Environment|Backup**.


Customize|Environment|Backup



2. 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 that 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 on 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, CodeWright 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 checkbox 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 (_). (See the topic *Transformation Patterns*.)

Examples:

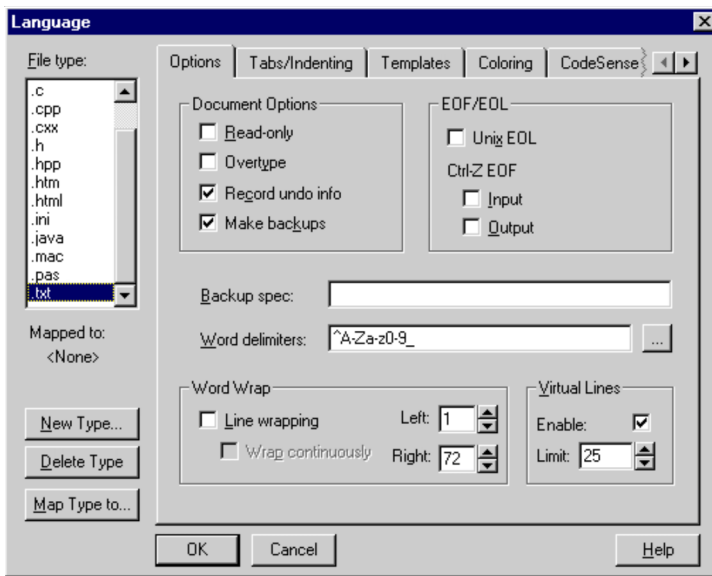
- ✓ The file FOO.C would be auto-saved under the name FOO.C_~.
 - ✓ A file whose name already has three characters, such as FOO.PAS, is auto-saved under the name FOO.PA~. In this case the S was replaced with a tilde.
 - ✓ A file with no extension, such as README, would be saved with the extension ._~ (e.g. README._~).
- The **Auto Save limit** listbox 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), access the **Options** tab on the **Language** dialog:

1. Select **Customize|Language| Options**.

Customize | Language | Options



2. In the **File type** box, highlight the extension of the desired file type.
3. 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).

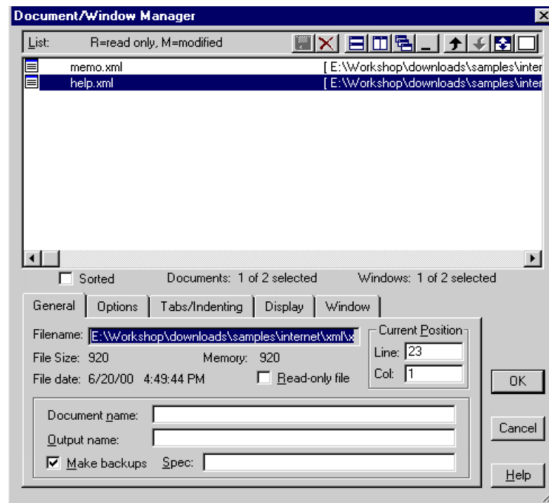
Refer to the topic *Formatting the Backup Specification*.

Backup Settings for Individual Files

To make a backup specification for an individual file (that overrides any global or file type specifications), access the **General** tab of the **Document | Manager** dialog.

1. Select the **Document | Manager | General** dialog.

Document|Manager|General



2. In the **Document List** box, select the document whose settings you wish to edit.
3. Check the box **Make backups**.
4. 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*.)

Formatting the Backup Specification

CodeWright 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 CodeWright 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.

Important: It is possible to specify a format that will result in an illegal filename. CodeWright 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.

To turn backups off, leave formatting strings empty at the global and local levels.

- ✓ 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 file C:\SRC\FOO.BAR:

Backup String Format Controls		
Control	Represents	Description
%b	basename	The entire name of the output file (fully qualified), less the extension. (e.g., C:\SRC\FOO)
%d	directory	Includes the path, less the drive (volume) specifier and filename. Ends with a backslash only if describing the root directory. (e.g., \SRC)
%[{... }]e	extension	Includes the dot that separates basename from extension, unless the name of the output file has no extension. May optionally contain a transformation pattern, described below. (e.g., .BAR if no transformation pattern supplied)
%f	filename	The root name and extension.
%p	path	Does not include the drive (volume) or filename. Ends with backslash. (e.g., \SRC\)
%[{... }]r	root	Does not include extension, drive (volume) or path. May contain an optional transformation pattern, as described below. (e.g., FOO if no transformation pattern supplied)
%v	volume	The drive letter and the colon. (e.g., C:)
%%	percent	A single percent sign is represented by two consecutive percent signs (%%).

Example: The formatting string, which represents the initial default, is %b.bak. You would set this default using a function call like:

```
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 is pointless, since the resulting string will be empty.

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 specified character replaces the character in the original string.
- 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 (i.e., 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 (i.e., 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:

Transformation Pattern		
Pattern	Original Name	Backup Name
%r%{??~ ___}e	FOO.CPP FOO.C	FOO.CP~ FOO.C_~
%r%{??~}e	FOO.CPP FOO.C	FOO.CP~ FOO.C
%r%{~??}e	FOO.CPP FOO.C	FOO.~PP FOO.~
%r%{~}e	FOO.CPP FOO.C	FOO.~ FOO.~
%r%{?}e	FOO.CPP FOO.C	FOO.C FOO.C
%{??????~ ~~~~~}r%e	FOO FOO.C TESTFILE.C	FOO~~~~~ FOO~~~~~.C TESTFIL~.C
%{?????}r%{??? bak}e	FOO FOO.C TESTFILE.C	FOO.BAK FOO.CAK TESTF.CAK

Making Files Read-only

Files can be assigned a read-only (non-editable) status at several locations within CodeWright, 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 (e.g. 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.

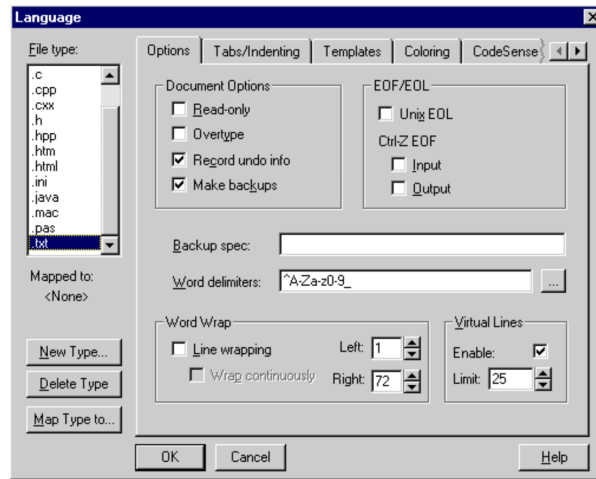
→**File|Open** – Directs CodeWright to open this file as read-only, overriding all prior settings.

File Types

To make files of a certain type read-only, access the **Options** tab of the **Language** dialog:

1. Select **Customize|Language|Options**.

Customize|Language|Options



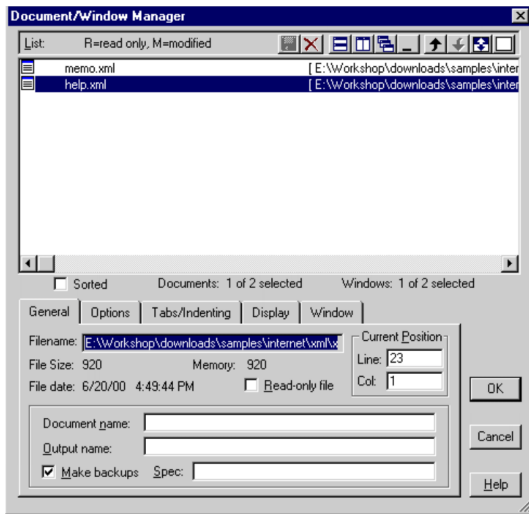
2. In the **File type** box, click and highlight the extension for the desired file type.
3. In the **Document Options** group, check the box **Read-only**. Files of the specified type may not be edited, unless this setting is overridden on the **Document|Manager|General** dialog, or the **File|Open** dialog.

Individual Files

To make an individual file read-only, complete the following steps.

1. Select **Document|Manager|General**.

Document | Manager | General



2. Click on a document in the **Document List** field to edit its settings.
3. Check the box **Read-only file** to prevent editing of this file, regardless of what has been previously specified for files of this type.

Individual File upon Opening

To make a file read-only upon opening it, regardless of prior settings for the individual file or the file type, complete the following steps:

1. Select **File | Open**.

File | Open



2. Browse to find the file you wish to open and click on the filename.
3. Check the box **Open as read-only**.

FTP: File Transfer in CodeWright

CodeWright offers FTP support (when enabled) under the **File** pull-down menu. The feature supports file structures for UNIX, Tandem Guardian, and VMS hosts. It can be used for loading files from any of these systems into CodeWright. The files can subsequently be transferred back when the editing is done.

To enable CodeWright's FTP support: complete the following steps:

1. Select the **Customize | Libraries** dialog.
2. Check the **FTP File Transfer** box in the list of **CodeWright Libraries**.
3. Click **OK**.

An **FTP** submenu is now available on the **File** menu. The submenu consists of:

- **Login** - Used to log in to an FTP host. The host must be supplied as well as the user name and password needed to access it. If the **User** and **Password** fields are left blank, one of two things may occur:
 - ✓ An “anonymous” user name and blank password will be used.
 - ✓ The user name and password corresponding to the host of a previously successful FTP login session will automatically insert in the **User** and **Password** fields.
- **Manager** – Used to manage files on remote systems and to perform FTP operations.
- **Put** - Used for standard Put operations.
- **Get** - Used for standard Get operations.
- **Get Files** – Used for getting one or more files.
- **Options** - Brings up the **FTP Options** dialog. Use the **Options** dialog for:
 - ✓ Setting the local and remote directories.
 - ✓ Looking at files in remote directories.
 - ✓ Setting permanent root directories.
 - ✓ Choosing whether local files to be 'Put' are the documents open in CodeWright, the files that make up a current CodeWright project, or any files stored on the local machine.
 - ✓ Omitting the prompt before you 'Get'/'Put' the current document.
 - ✓ Requesting an audible 'beep' when file transfer is complete.
 - ✓ Setting whether the files to be 'Put' should be in ASCII or Binary.
- **Disconnect** - Disconnects the current connection.

15- Large Files

CodeWright can accommodate files as large as 2 Gigabytes. Any file over 1 MB is considered a large file. There are several ways to configure CodeWright to make it more stable and efficient when working with large files. Among these are:

- Manipulating swap blocks.
- Turning off backup files.
- Turning off scroll bars.
- Preloading files.
- Turning off ChromaCoding.
- Using the File Rewrite Save Method for saving files.

Swap Blocks

CodeWright uses swap blocks to manage the memory allocated for an open, active file. CodeWright 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, CodeWright 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 CodeWright.

Example: The default for swap blocks is 100 8K-swap blocks. You may want to try increasing this (the maximum is 1000).

How to Change the Number of Swap Blocks

Set the number of swap blocks in your configuration file (CWRIGHT.INI) located in the CodeWright home directory. Find the [Editor] section of this file and insert a line under that heading similar to the one shown below, then restart CodeWright 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 CodeWright'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 CodeWright.
- 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 should be used from CodeWright's shortcut command line, a DOS command line, or any of the Sync program command lines.

- To modify the command line for the CodeWright shortcut:
 1. Right-click on the shortcut and choose **Properties**.
 2. Click on the **Shortcut** tab.
 3. In the **Target** edit box, add **-BlockSize=0x8000** after the CW32.EXE.

Example: C:\CW32\CW32.EXE -BlockSize=0x8000

- Do the following to run the "**-BlockSize=**" from any of the synchronization configuration dialogs:
 1. Run the **Synchronization Configuration** dialog for a synchronized environment. The synchronization configuration settings are accessed in VCSync.exe for MSVC, and under the **About CodeWright Help** menu item in the synchronized environment associated with other sync programs. See the chapter *Synchronization*.
 2. After the full path name for CodeWright, e.g., "C:\CW\CW32.EXE", add the **-BlockSize=** parameter.

Example: C:\CW32\CW32.EXE -BlockSize=0x8000

The 0x8000 setting can be any setting within the limits described above.

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. Short of that, if CodeWright is the only application you are using, you can be pretty generous with swap blocks.

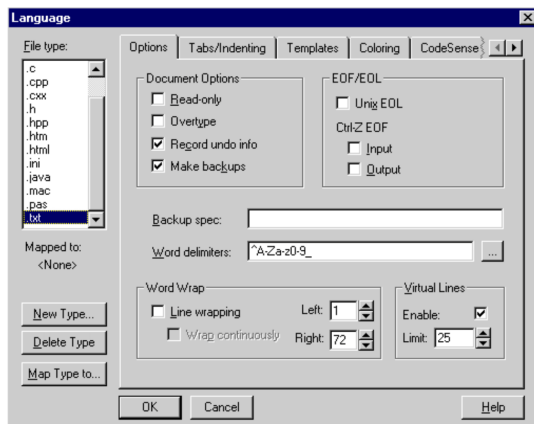
Backup Files

You may not wish to have CodeWright 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:

1. Choose the **Customize|Language|Options** dialog

Customize|Language|Options



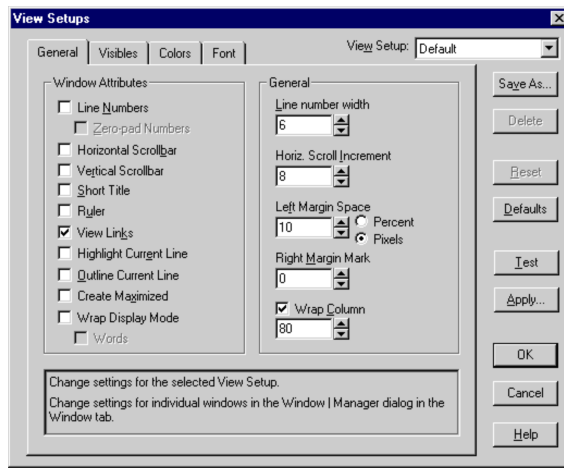
2. In the **File type:** box, select the file type(s) for which to turn off backups.
3. Unmark the box labeled **Make backups** in the **Document Options** group.

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:

1. Select **Customize|View Setups|General**.

Customize|View Setups|General



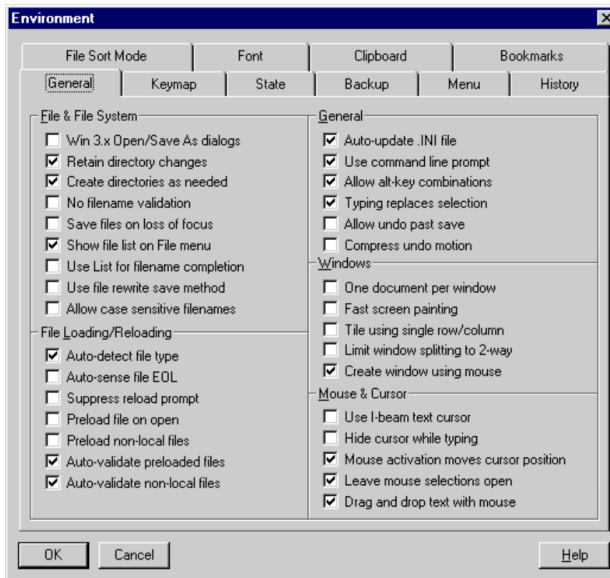
2. In the **Window Attributes** group, make sure that the **Vertical Scrollbar** and **Horizontal Scrollbar** boxes are unchecked.
3. Click **OK**.

Pre-loading Files

When CodeWright pre-loads 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 CodeWright's feature for pre-loading files. Make sure the options are not marked if you want CodeWright to handle large files more efficiently. To disable the preloading options:

1. Select the **Customize|Environment|General** dialog.

Customize | Environment | General



2. In the **File Loading/Reloading** section, make sure the options **Preload file on open** and **Preload non-local files on open** are unchecked.

When the options are turned off, instead of loading the whole file into memory, CodeWright only loads as much of the file as necessary for editing and maintains pointers to the original file. See the chapter on *File Loading, Backup and FTP* for more information.

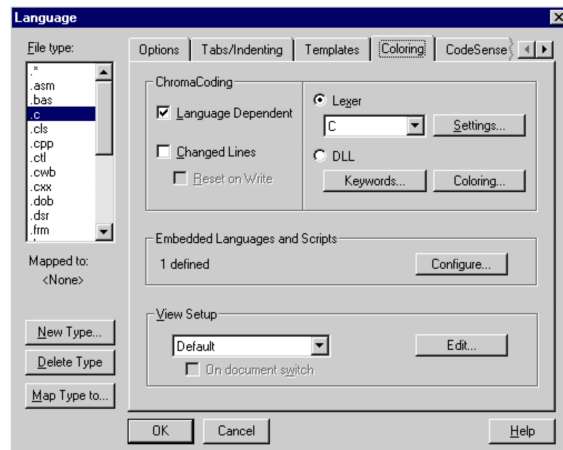
Turn off ChromaCoding

If speed is of the utmost importance when loading large files, you could optionally turn off CodeWright's ChromaCoding, the feature that colors programming language syntax structures.

To turn off ChromaCoding:

1. Select the **Customize | Language** dialog.
2. Select the file type being edited in the list of **File Types** on the left.
3. Click on the **Coloring** tab.

Customize | Language | Coloring



4. Uncheck the following features in the **ChromaCoding** section:
 - **Changed Lines**, which causes CodeWright to color lines that have been changed.
 - **Language Dependent**, which causes CodeWright to color files according to the syntax of a particular programming language.

Saving Large Files

CodeWright's 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 CodeWright uses to open and save files works in the following way:

Given a file, FOO.C

- Open FOO.C into CodeWright.
- CodeWright 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 CodeWright 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:

Example: Given a file, FOO.C

- Open FOO.C into CodeWright.
- 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.000) 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 haven't been modified won't be rewritten, reducing the time needed to save the file.

To turn on the File Rewrite Save Method:

1. Select the **Customize|Environment| General** dialog.
2. In the **File and File System** section, put a checkmark in the **Use file rewrite save method** option.

Chapter 16

16- Extend CodeWright

CodeWright is a fairly complete product. There is not much that it lacks. Nevertheless, there always seems to be one job for which necessary tools just can't be found. If this is the case, CodeWright has some tools for extending the CodeWright program.

CodeWright's extensibility tools consist of three macro languages (Perl, AppBasic, and API Macros), and DLL-source code (CodeWright DLLs, or Add-Ons, can be loaded interactively from the **Customize | Libraries** dialog). The tools are provided in the event that CodeWright doesn't already have the necessary functions for handling the task at hand. They can be used together (i.e. a macro that uses a function from a DLL or another macro, or vice versa), or on their own.

- ✓ The advantage of using macros is that they can be used on the fly, without the added complexity of compiling.
- ✓ The advantage of using DLL Add-Ons is that they are more flexible, and can be written in any programming language that is capable of creating DLLs.

This chapter first describes some of the more technical aspects of using the CodeWright API from CodeWright's Command Key. It then discusses the advantages of using Macro languages or DLLs to extend CodeWright. Finally, it goes into detail about CodeWright's three macro languages (Perl, AppBasic, and, API Macros) and describes some of the CodeWright architecture as it pertains to the construction or modification of CodeWright DLLs. The preliminary components of a CodeWright DLL are also described along with some tips for compiling the DLLs.

The assumption going into this chapter is that the user has some familiarity with CodeWright. In particular, it is assumed that there is some familiarity with the CodeWright API, and the methods used to assign API functions to keystrokes, buttons, and/or menu items.

CodeWright API

CodeWright has a multitude of API functions that can be used interactively from within the interface (i.e. from **Tools | API Command**, or attached to menu items, buttons, or keystrokes). APIs can also be used in custom macros and DLLs designed for extending CodeWright. Various chapters in this manual have discussed the process of using APIs from within the CodeWright interface.

The remainder of this chapter discusses the use of CodeWright APIs from the Command Key (**Tools | API Command**) more extensively, and it elaborately describes the use of the APIs in macros and DLLs. Help and examples for most CodeWright APIs can be accessed in CodeWright's online help.

Using the API from the Command Key

CodeWright's **Command Key** was described earlier in this manual, in the chapter on *Command Key, Libraries, & Environment*. This section will describe some of the more technical aspects of using the Command Key. Remember that the **Command Key** is **F9** if you are using the CUA or 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** will always access the prompt.

To use the CodeWright API, you need to at least know the name of a function you want to execute. CodeWright's online help is the place to learn more about the CodeWright API and specific functions.

If you are trying to avoid learning any more about CodeWright's API than you have to, make sure you have checked various dialogs and menu entries to see if what you are trying to do can be done through a menu.

Displaying Return Values With the Command Key

Many CodeWright 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.

Here is an example of how such a command might look, just before sending the command off for processing:

```
?BufQModifiedCount
```

When the return type is a numeric value, CodeWright displays that number on the status line in both decimal and hexadecimal notation. If the return type is a pointer to a string, CodeWright displays the contents of that string.

Run Multiple Commands

It is possible to use the Command Key to execute multiple functions and expressions at once. The functions and expressions can be nested or separated by commas. They must be preceded by an asterisk (*) and surrounded by parentheses.

Example: When run from the Command Key, the following nested functions will find and attach the buffer 'test' to the current window, if 'test' exists.

```
* (WinAttachBuf (WinQCurrentWindow() ,  
BufFindBuffer ("test")) )
```

The asterisk causes functions and expressions that are inside the parentheses to be passed to the CodeWright API **MacroExecMultiple**.

If the asterisk is also preceded by a question mark (?), the results of the expression will be displayed on CodeWright's status bar.

Examples of Command Key Usage

All three of the examples below do the same thing. They put CodeWright in the Hex Editing mode, with the cursor located in the "binary" portion of the display.

```
■ BufSetHexBinary  
BufSetHexBinary()  
bufsethexbinary
```

The examples below advance the cursor position by two characters.

```
■ MovNextChar( 2 )  
Movnextchar 2
```

The commands below will open a document for C:\PROJECTA\DEBUG.C.

```
■ BufEditFile( "c:\projecta\debug.c" )  
bufEditFile "c:\projecta\debug.c"  
BufEditfile=c:\projecta\debug.c
```

The following example assigns the function BufSetHexAscii to the keystroke



```
■ kmapAssign( "<ctrl-a>", "BufSetHexAscii" )
```

The example below begins an inclusive selection, using one of the labels listed in CodeWright's online help.

```
■ MarkBeginSel( SELECTION_INCLUSIVE )
```

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. CodeWright 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:

Operator	Operation
()	Grouping
-	Unary Minus
~	Bitwise Complement
!	Logical NOT
/	Division
*	Multiplication
%	Modulus
-	Subtraction
+	Addition
<<	Bitwise Shift Left
>>	Bitwise Shift Right
>	Greater Than
<	Less Than
<=	Less Than or Equal
>=	Greater Than or Equal

Operator	Operation
==	Equivalence
!=	Non-Equivalence
&	Bitwise AND
^	Bitwise Exclusive OR
	Bitwise OR
&&	Logical AND
	Logical OR

For many users, using CodeWright's API functions from the **API Command Key** is the first step toward truly customizing and extending CodeWright. Once users see how quickly they become comfortable with CodeWright's API, they want more. Soon they start using them in CodeWright macros and custom DLLs and subsequently delving into the supplied DLL source code for more. Read on to find out how to get the most out of the CodeWright API by using them in CodeWright's macro languages and DLLs.

Macros, Macro Languages and DLL's

The most powerful way to customize CodeWright is through the use of its three macro languages, and by creating and customizing its DLLs. The following sections discuss and compare these methods for expanding the CodeWright program.

Macros and Macro Languages

A macro might be defined as something simple that represents something bigger or more complex. Under this definition, macros include everything from keystroke recordings that are assigned to keys, and the % macros CodeWright uses in command lines, up to more sophisticated things written in programming languages.

A Macro Language provides a method for creating macro source code, which may contain control structures and variables. Macros are normally interpreted at runtime. CodeWright has three macro languages for extending its program. The table below describes some pros and cons for the languages. The languages themselves are discussed in the sections *Perl*, *AppBasic*, and *API (C-like) Macros*. Read on to determine which language is best for you.

Macro Language	Strengths and Weaknesses
Perl Macros	<p>Perl, along with JavaScript, is perhaps the most popular language for writing extensions to web pages. The syntax is similar to C or AWK, but it has many built-in features for the Internet.</p> <p>Perl is least suitable for writing simple functions for assignment to keys because of the time it takes for the interpreter to initially load. It does, however, allow Perl programmers to program in a familiar language, using familiar extensions and libraries. CodeWright's version of Perl is based on the Gnu-released Perl interpreter.</p>
AppBasic Macros	<p>AppBasic is CodeWright's 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 CodeWright API functions, except those designated as non-interactive. ■ Functions in most any external DLL. <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>CodeWright's 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 Tools API Macros dialog, or you can write them in one of CodeWright's standard edit windows.</p>

DLL Extensions

You can write extensions for CodeWright using any compiler that can produce DLLs. We refer to this capability as DLL Extensibility. Most sophisticated extensions are written using DLL Extensibility. This method of extending CodeWright has the following benefits:

- Familiar compiling, and debugging tools,
- Unbeatable speed, and
- Access to functions in other DLLs and libraries.

For details on DLL Extensibility, refer to the topic *Making DLL Add-Ons*, in this chapter.

DLL Extensibility, however, may not be as convenient as other methods of extending CodeWright:

- It is not as convenient 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, which doesn't come with CodeWright.

Keep these in mind as you consider extending CodeWright using Dlls or one of the supplied macro languages (Perl, AppBasic, and API).

Where is it Defined?

Once an extension has been created (whether the extension is a macro or a DLL), there is an opportunity for confusion as to exactly where the function being executed is coming from. It could be a CodeWright built-in function, a CodeWright DLL function, an AppBasic macro, a Perl sub, a keystroke macro or an API macro. The command **LibFunctionExistsWhere()** helps resolve the ambiguity.

The **LibFunctionExistsWhere** command returns an allocated string identifying the location of the function name that is supplied as its argument (0 is returned if it doesn't claim the function). Listed below are a few sample 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, i.e. 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**.

Perl

This section describes CodeWright's Perl macro language.

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 CodeWright, you can make use of CodeWright custom facilities that reinforce these views as needed. They are described under the topic *Loading and Running Scripts*.

If you have not already installed a command line version of Perl, we have included a copy of the ActiveWare 3.15 build of Perl 5.0 for Win32, PW32I315.EXE in the PerlW32 directory on the CW CD.

Getting Started with Perl

There are two loadable DLLs necessary to use Perl with CodeWright: the Perl interpreter and the Perl language support DLL. To enable these modules, go to **Customize|Libraries** and select two checkboxes: **Perl Extension Language Interpreter** and **Perl Language**.

Note: Updating your version of Perl should not pose any problems, nor have any effect on the CodeWright Perl interpreter. The CodeWright Perl interpreter has been modified for use with CodeWright, and is completely independent of any other Perl interpreter that may be installed on your system.

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** checkbox to enable the Output Window's **Perl** tab.

The CWPDLL is a Perl extension module that provides access to CodeWright API functions from within Perl scripts. In general, you can use any of the CodeWright API functions listed in the online help, except for those whose data types are incompatible with Perl. For a list of available CodeWright API functions, see the contents of the CWPPM file.

Creating and Editing Perl Scripts

Perl scripts can be created and edited in standard CodeWright edit windows. There are some sample Perl script source files in CodeWright'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)";
```

Supplied Perl Macros

CodeWright comes with a number of sample Perl macros that can be used as starting points for creating custom macros. The macros are contained in files with .PL extensions. They are located in the MACROS subdirectory of the CodeWright installation directory.

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. You will find this setting in the **Tools|Perl Macros| Properties** dialog.

To help distinguish **stdin**, **stdout**, and **stderr** elements of the Output Window, CWPPerl 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, for the view setup labeled **Output Window**.

Popup 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 **Tools** menu, check out this menu. The **Properties** item on the popup menu brings up a dialog that lets you control how Perl interacts with CodeWright. 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 CodeWright 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 CodeWright'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  . 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 CodeWright 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> consist of any parameters required by the script file.

This will load the script, execute it, and unload it 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 Perl macros supplied by Starbase by checking the appropriate box in the top of the dialog in the area called **CodeWright macros**.
 - ✓ Any Perl macros stored in the CW32\MACROS directory will show up in the CodeWright 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 **CodeWright 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 CodeWright API command:

```
PerlLoad <filename>
```

Once the macro is loaded, you can invoke it by going to the CodeWright **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. Select the desired subroutine and press **Run**.
4. Enter parameters at the resulting prompt, or hit return to run without them.

Executing a Script

You can load and execute a Perl script from CodeWright's API Command Line by using the command:

```
PerlExec filename.pl
```

If you only want to execute a specific subroutine in a script use:

```
PerlExecSub filename.pl subname
```

In both of these cases, you may include additional 'command line' parameters. All text following the syntactical elements shown in the two commands is broken down into 'parameters' using CodeWright's normal command line parsing rules, with respect to spaces, quotes, escapes, etc.

If you just want to execute a simple Perl script that can be typed in one line use:

```
PerlExecStr 'Perl-string'
```

Interpreters

Perl for CodeWright 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 may 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 thusly:

```
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 CodeWright Functions from Perl Scripts

Over 800 CodeWright functions can be directly accessed from within Perl scripts in CodeWright. To do so, the following line must be placed at the top of the 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 CodeWright 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 CodeWright 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 CodeWright's **BufQCurrentLine** API in the following manner:

```
$line = CWP::BufQCurrentLine();
```

Constants that are held in CodeWright's lookup table can be accessed using a special function. For example, to see if there is a column selection present you could use:

```
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 CodeWright 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 Perl's Namespace

If you have CodeWright functions that you access frequently and their names don't conflict with standard names, you can save some typing by telling Perl that you want certain names imported into Perl's namespace.

To import names into Perl's 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

```
PerlUnload filename.pl
```

Using Perl's 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 CodeWright, you accomplish the same thing by checking the **Debug Mode** checkbox in the **Perl Properties** dialog (right-click the **Perl** tab).

When running a script in debug mode, there are three important requirements:

- You should only use debug mode when invoking scripts with **PerlExec** on the API command line.
- The script must not already have been loaded by CodeWright 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 merely by typing their names on the CodeWright API Command Line. CodeWright has a mechanism to allow Add-Ons to respond to indicate their recognition of otherwise unknown (to CodeWright) 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 simply by typing **MyPerlFunc** at the CodeWright 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. CodeWright 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 800 or more functions available from the CodeWright 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 CodeWright 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 CodeWright function **EvalExpression**.

■ **DWORD CWExec(LPSTR cw_funcname, ...);**

This function provides an alternate form of executing a CodeWright function that has been made available via **LibExport()**, i.e. all functions provided by DLLs, both standard and Add-On. The advantage to using this form, rather than **LibFunctionExec()**, is that you needn't 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_SCRAP 0x0004 // input from scrap buffer  
#define PERL_OUT_CON 0x0000 // output to 'console'  
#define PERL_OUT_BUF 0x0001 // output to current buffer  
#define PERL_OUT_SEL 0x0002 // output to selection  
#define PERL_OUT_CLIP 0x0003 // output to clipboard  
#define PERL_OUT_SCRAP 0x0004 // 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 CodeWright

These files are found in the CodeWright home directory after installation:

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 the Output Window.

These files are found in the CodeWright PerlLib subdirectory after installation:

Filename	Purpose
CWP.DLL	Allows access to CodeWright API's in the Perl script.
CWP.PM	Exporter for CodeWright API's.
CARPPM	Error routines.
CONFIG.PM	Win32 Perl configuration.
CWPXS	Source Perl script used to rebuild CWP.DLL.
DYNALoader.PM	Dynamically loads C libraries into Perl code.
EXPORTER.PM	Default import method for modules.
PERL5DB.PL	Perl 5.0 Debugger.
TYPEMAP	CodeWright variable types (for use with CWPXS).
XSUBPP.PL	Converts Perl XS code into C code (for use with CWPXS).
TERM\CAPPM	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*. Some useful URLs would include <http://www.perl.org> for general Perl information and <http://www.activestate.com> for Perl for Win32 systems.

If you want certain .PL files to be accessible to CodeWright (e.g. a math library), put them in the `perlib` subdirectory of the CodeWright home directory, or specify a path to them on the **Tools|Perl Macros|Properties** dialog. For the latter, make sure the path you specify does not contain .PLL files needed by CodeWright's Perl Interpreter.

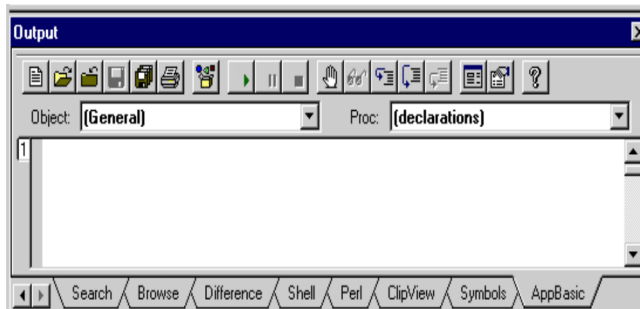
AppBasic

This section explains how to use the AppBasic Macro Language in CodeWright to create your own macro functions.

CodeWright's 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, CodeWright's API, and functions in independent DLLs. Example files are supplied in CodeWright's *MACROS* subdirectory to aid you in getting started. 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 CodeWright's Output Window, you can add break points and debug your macro as well.

Output Window with AppBasic Tab Selected

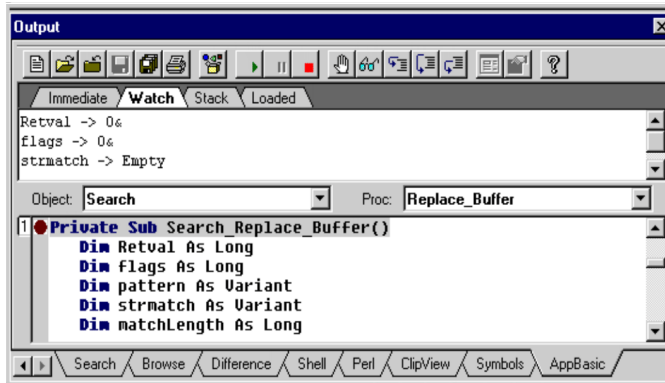


Getting Started

To enable the AppBasic Macro Language, go to the **Customize|Libraries** dialog. You will see a list of optional packages or modules you can use with CodeWright. Select the checkbox for **AppBasic Macro Language**. There is also a **Basic** module listed that provides template expansion and language coloring support for the Basic programming language in general. We suggest you enable this option as well.

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.

AppBasic Environment



























Two Editors





You can select one of two editors to use to edit your AppBasic source code. You can use CodeWright as your editor, or the Rich Text Editor.

- If you use CodeWright, 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 CodeWright 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.

Keystroke	Operation
 + A	View Macro
 + E	View Immediate Tab Window
 + W	View Watch Tab Window
 + T	View Stack Tab Window
 + L	View Loaded Tab Window
	Debug Run
 + 	Debug Stop
	Debug Pause
	Debug Step Into
 + 	Debug Step Over
 + 	Debug Step Out
	Debug Step To
 +  + 	Debug Clear All Breakpoints
 + 	Debug Add Watch
 + 	Debug Quick Watch
	Debug Toggle Breakpoint
 + N	File New

Keystroke	Operation
 + O	File Open
 + -	File Close
 + S	File Save
 + P	File Print

Two Toolbars

AppBasic has two nearly identical toolbars that you can use with it. 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 CodeWright's Dockable Toolbars. You can position it anywhere on your screen, or dock it against any edge of the CodeWright 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.

Popup Menu

When you right-click in the AppBasic 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, check out 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. CodeWright 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:

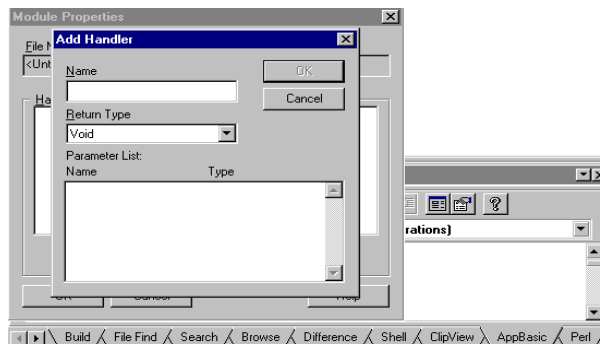
1. Select the **AppBasic** tab on the Output Window.
2. Right-click in the window and select **Properties**. In the **Module Properties** dialog, you can provide the names of the AppBasic functions you want to call from **LibFunctionExec**, the **CodeWright API Command**, or bind to a CodeWright button.

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 CodeWright, Handlers serve the same purpose as calling **LibExport**. Once made available to CodeWright in this way, Handlers can be assigned to keys, run from menus and the like.

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 CodeWright.

3. When you have finished adding your functions, select **OK**. Then, on the **Modules Properties** dialog, select **Add**. The **Add Handler** dialog displays.

Add Handler Dialog



Creating a Handler

To create a Handler:

- In the **Add Handlers** dialog, name your function and declare its type.
 - Note:** Functions and Subroutines must start with a letter and can then be followed by an underscore or letter.
- Type in any parameters you wish.
 - Note:** 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 CodeWright 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 CodeWright menu **Tools** | **API Command** and type:

DoThis

10. Click **OK**. A message box should appear with the words "This is a test" in it.

Note: It is not always expedient to load a 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 Drop-down Lists

The **Object** list shows all the objects for the current module. The object named (*general*) 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.

The Object and Proc Drop-downs located below the toolbar



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*.

Example: 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()**. An example is provided next.

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

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 checkbox in front of the filename will unload the macro. Calling **cwbUnloadFile** (filename) through the API Command prompt will also unload it. 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.

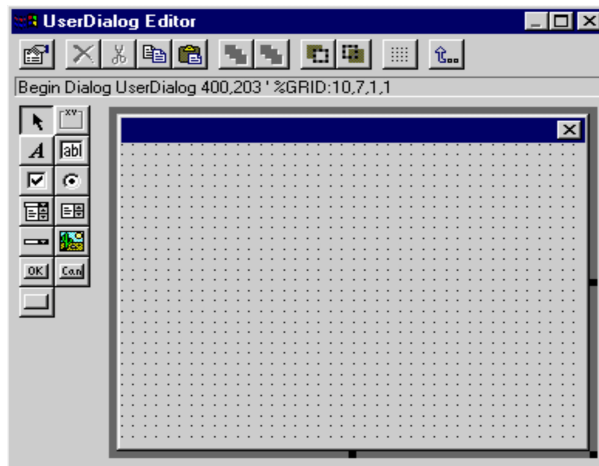
Creating a Modal User Dialog

To create a modal user dialog:

1. Place the cursor in the subroutine or function in which to place the dialog.
2. Then either:
 - Select the AppBasic toolbar button **Edit UserDialog, OR**
 - Select the CodeWright AppBasic toolbar button **Insert/Edit User Dialog**.

The **UserDialog Editor** displays:

UserDiaog Editor



3. Select the **Edit Properties** toolbar button (or right mouse click on the dialog that you are creating). The **Edit UserDialog Properties** screen displays.

Edit UserDialog Properties

The screenshot shows a dialog box titled "Edit UserDialog Properties". It has several input fields and checkboxes. The "Left" field is set to 0, "Top" to 0, "Width" to 400, and "Height" to 203. The "Caption" field is empty. The "Dialog Function" field is empty. The "Comment" field is empty. There are checkboxes for "Centered" (checked) and "Quoted" (checked). There are three buttons: "<<", ">>", and "Close".

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.

Note: 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. Hit 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.

Note: 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 on 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:

```
Dialog dlg
```

To:

```
bButtonPushed = Dialog(dlg)
```

The final sample is listed below:

```

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 CodeWright 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.

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 CodeWright's 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 CodeWright's API functions, writing your own event handler gives you access to CodeWright's core. You can change the way critical functions work without rewriting CodeWright.

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 CodeWright API Library 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 CodeWright 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")
```

Note: De-registration of the event is handled automatically when the macro file is unloaded in CodeWright. To stop an event handler before terminating the program (module) set the returned EventHandler Object to "Nothing".

Additional information on events can be found in the CodeWright API Library help file under the section *Programming* and the topics *Events* and *Using Events*.



Refer to the sample file `EHTEST.CWB` that follows.

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:

- Open the macro file for modification, if you have not already done so (load it into the AppBasic Window).
- Set a Break point, perhaps at the first line of the Sub or Function you are debugging, by pressing  or clicking the **Toggle Breakpoint** button. (A dot should appear in the margin to the left of the line.)
- Enter *Run* mode by clicking on the **Run**, **Step Over** or **Step Into** button. The **Immediate**, **Watch**, **Stack** and **Loaded** tabs will then appear above the Edit Window. You can also press  to enter this mode if you are using the RTF editor, instead of the default CodeWright edit window. (**Tools | AppBasic Macros | View | Rich Text Editor**)
- 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.

Note: 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, double-clicking in the Search Window will be disabled until you have stopped the macro.




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.

Command	Results (when you press )
?<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 Immediate Window 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.

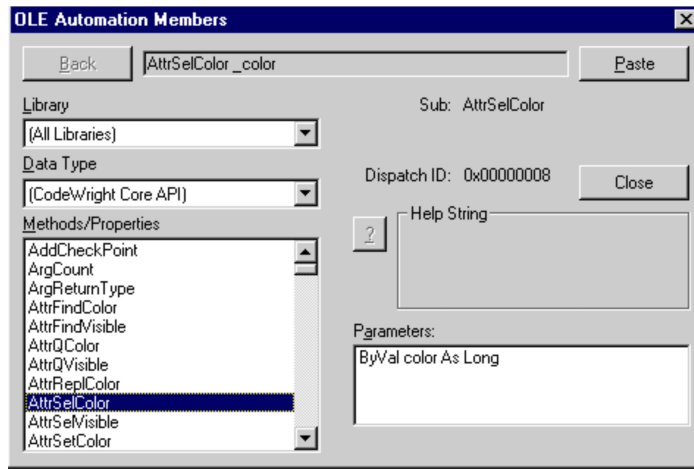
In addition, the following actions are available to you:

- The expression to the left of the "->" may be edited.
- Press  to update all of the values displayed, to reflect any changes you have made.
- Press   to delete the line.

Object Browser

The Object Browser shows information about all the special data types that are available.

OLE Automation Members



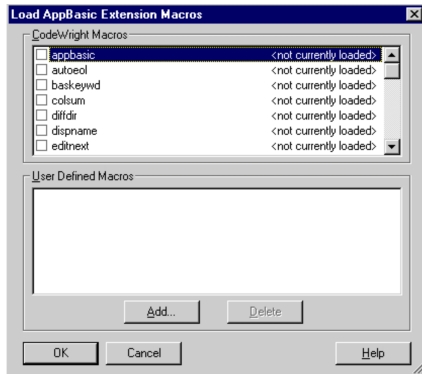
You can get to the Object Browser by pressing any of the following:

- 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 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, which allows sample macros or user defined macros to be loaded.

Load AppBasic Extension Macros



AppBasic Sample Macros

AppBasic macros are files that end with .CWB extensions.

Example: BASKEYWD.CWB

A number of sample macros are supplied in the \MACROS subdirectory of the CodeWright installation directory. Most of these macros are not intended to perform necessarily useful tasks, but rather are intended to show how features of CodeWright are accessed through AppBasic. The comments in the macros describe their functionality.

AppBasic-related API Commands

The following useful API commands are available from the CodeWright 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.

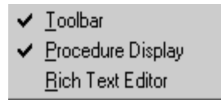
■ **cwbUnloadFile (<moduleName>)**

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 CodeWright Edit window to the WinWrap Rich Text Editor.



This configuration can also be set by right-clicking over the AppBasic Window, and then selecting **View** from the popup menu.

If you wish to change the configuration of the window programmatically, the following CodeWright API commands will assist you. These commands may be issued via 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 CWRIGHT.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 CodeWright Editor and the other is a Rich Text Editor.

Example: To change to a CodeWright Editor call:

```
cwbCreateCWWindow( 1 ) or cwbCreateCWWindow( TRUE )
```

Example: To change to Rich Text Editor call:

```
cwbCreateCWWindow( 0 ) or cwbCreateCWWindow( FALSE )
```

Example Configuration File Settings

The AppBasic window's configuration is stored in the CWRIGHT.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.

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

■ **cwbShowToolbar(int nShow)**

Hide/Show Standard AppBasic Toolbar.

■ **cwbShowProcDisplay(int nShow)**

Hide/Show Object and Proc Dropdown Lists.

API (C-like) Macros

This section covers CodeWright's API macros.

CodeWright API Commands are function calls that can be made interactively. This means that they may be assigned to keys, menu items, buttons, popup context menus, and issued from the API command line. CodeWright API Macros build upon this capability to provide a quick and simple way to write extensions for CodeWright.

If you have previously used, or read about using API Commands interactively, a few 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.
- Commas must separate parameters.

API Macros Defined

API Macros are one or more sequential, iterative, or conditional statements that are given a name. These statements often contain CodeWright API function calls. They are stored in a file, CWRIGHT.MAC, and may be recalled by name for execution within CodeWright.

A number of API macros are supplied with CodeWright. The supplied macros can be viewed in one of the following ways:

- Select the appropriate macro from the drop down list under the **Name** edit box in **Tools|API Macros**. The macro is then displayed in the dialog for viewing or editing purposes.
- Open the file CWRIGHT.MAC into CodeWright, then locate a desired macro for viewing or editing.

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:

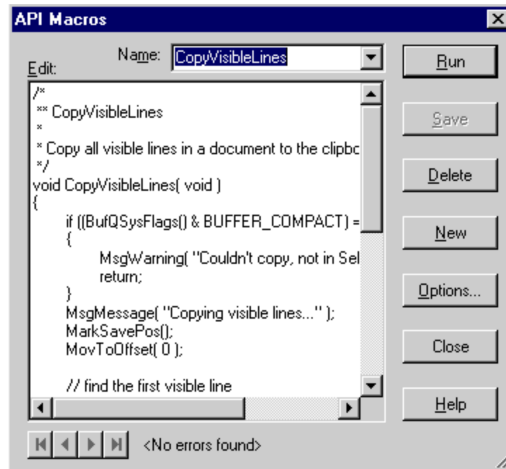
- Load the CWRIGHT.MAC file into CodeWright.
- See it correctly ChromaCoded.
- Compile it with the menu or button.
- Parse errors from the **Build** tab of the Output Window.

For more information on Lexers, refer to the topic *ChromaCoding Lexers* in the chapter *View Setups and Language Support*.

Getting Started with API Macros

The **Tools | API Macros** dialog lets you create, edit and test API Macros.

API Macros



Creating a Macro

To create an API macro in the **API Macros** dialog:

1. Give the macro a name in the **Name** edit box.
2. Type in the API commands you wish to execute into the **Edit** box.
3. Press the **Save** button. Press the **Run** button to 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. Press either **Run** or **Save** to save your changes to disk. **Run** will also run the macro.

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 **SHIFT** **CTRL** **↓**. To insert a <Tab> character use **CTRL** **TAB**. As in a CodeWright window, hit **F1** to bring up help for the function under the cursor.

Editing API Macros in a CodeWright Window

You can create and edit API Macros in a standard CodeWright edit window. This gives you access to CodeWright's advanced editing capabilities. Complete the following steps:

1. Open the file CWRIGHT.MAC into CodeWright.
2. Create a new macro using the following syntax:
[API Macro:<macro_name>]
-Or-
 1. 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.
 2. Make any necessary edits.
 3. Save before running the macro, since macros are always read from the file.

Running a Macro

The obvious way to run a macro is from the menu or dialog. Macros can also be executed from within CodeWright in as many ways as API functions can be. Just use the macro's name, and optionally its parameters, like CodeWright API functions.

Language Definition

For more information on the API Macro language, see the following discussions on:

- Comments
- Identifier Naming Rules
- Data Types for Variables
- Declaring Variables and Arrays
- Literal Values
- Array Initializers
- Automatic Type Conversion
- Expressions
- Statements and Statement Blocks
- Program Flow of Control Structures
- CodeWright Event Handling
- Difference between API Macros and C
- String Functions

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.

Data Types for 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 CodeWright's API commands. These include `'char'`, `'int'`, `'long'`, `'UINT'`, `'WORD'`, `'DWORD'`, `'BOOL'`, `'HBUFFER'`, `'HWINDOW'`, 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 (`"`).

Arrays of either type (int or string) are also supported. An array is a variable capable of storing one or more values. All of the values in the array must be of the same type. The values stored in the array are called `'array elements'`.

Declaring Variables and Arrays

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 haven't had a value assigned to them, their value will be zero or NULL.

Arrays can be declared using one of the following forms:

- `<arr_name>` is the name of the array and must conform to identifier naming rules.
- `<arr_size>` is optional. If given, `<arr_size>` must be a constant expression that evaluates to a nonnegative integer value.
- If `<arr_size>` is given and the declaration does not include an assignment, all array elements are initially set to 0 (NULL for string arrays).
- If `<arr_size>` is not given, the size is initially zero. Omitting `<arr_size>` is allowed because array size is dynamic and will change as elements are assigned to the array.

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 Types

Declare parameter variables and return types 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 defining parameter variables and return types:

- For backward 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.
- Arrays defined for parameter and return types must be automatically sized (use no number within the brackets for the array size).
- 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 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 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:

Literal	Escape Sequence	Value
\0	NULL	0x00
\a	Alert (bell)	0x07
\b	Backspace	0x08
\f	Form Feed	0x0c
\n	New Line	0x0a
\r	Carriage Return	0x0d
\t	Horizontal Tab	0x09
\v	Vertical Tab	0x0b
\xhh	Hexidecimal Value	(00-ff)
\\	Backslash	

Note: 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.

Array Initializers

Array initializers for API Macros take the form:

```
{<element_0>,<element_1>,...<element_n>}
```

- For arrays that have many elements, the '...' represents the repetition of the pattern '<element_1>, <element_2>' up to <element_n>. The ellipsis cannot be used explicitly.
- There may be zero or more elements in an array initializer.
- Any valid expression can be used as an element in an array initializer. However, character, number or string literals are commonly used.
- Array element types need not match the array variable that the array initializer is used with. Types are converted as needed.

Array literals are valid operands and can be used in expressions with other operands.

Example: The following is an example of combining an array declaration with an array initializer:

```
int iArr[10] = {1,2,3}
```

In special cases like the previous example, where the array initializer has fewer elements than the declaration size, the remaining elements are initialized to 0 (NULL for string arrays). Otherwise, when an array type is assigned to an array variable, the array variable takes the size of the array being assigned to it.

Automatic Type Conversion

Values are automatically converted to another type when used where that type is expected.

Type Conversion Rules		
Expected type	Given type	Rule
int	string	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 or NULL.
string	int	Convert value to decimal characters.
int array	string array	Convert string elements to int elements.
int array	int	Make an array of size one with the int at index zero.
int array	string	Make an array with the same length as the string using each character from the string for array elements.
string array	int array	Convert int elements to string elements.
string array	string	Make an array of size one with the string at index zero.

Type Conversion Rules		
Expected type	Given type	Rule
string array	int	Make an array of size one with the int converted to a string at index zero.
int	int array	Sum all elements.
int	string array	Convert to int array then sum all elements.
string	string array	Working from first to last, append each element onto the string made from the preceding elements.
string	int array	Make a string with the same length as the array using a character from each array element.

The last two conversion rules have a special option that allows the appended string elements to be separated by a given string.

Use the API macro built-in function **joinStr** to specify and/or query this string.

Examples:

- Use a single space between each joined (appended) string element.

```
joinStr(" ");
```

- Query the current string used to join elements.

```
retStr = joinStr();
```

Expressions

Expressions are made up of operands and operators. Operands can be variables (including arrays), array initializers, literals, and function or macro calls. Support is provided for all C operators. Numeric, logical and string expressions are also allowed as operands within larger expressions. Expressions can be used as function call parameters.

Accessing Array Elements within Expressions

Use this format to access an individual element within an array:

```
<arr_name>[<index_expression>]
```

- `<arr_name>` must be the name of an array or string variable.
- `<index_expression>` is an expression used to specify the index of the element.
- The first element of an array has the index of zero.
- When `<arr_name>` is of type string, the element accessed will be the character at the given index. Accessed string elements are converted to type int.

Example: The following shows how array elements can be accessed:

```
// define 2 int arrays, one of size 0 and one
// size 10
int iArr1[], iArr2[10];
// assign iArr2 to iArr1. iArr1 now has 10
// elements (all zero values)
iArr1 = iArr2;
// assign 2 to the first element of iArr1
iArr1[ 0 ] = 2;
// assign 1 to the second element (index 1) of
// iArr2
iArr2[ iArr1[ 0 ] - 1 ] = iArr1[ " " ] - 1;
// assign 1 to the third element of iArr2
iArr2[ { 2, 1, -1 } ] = { 1, 0 };
// Note the last example involves array type
// conversions. See the topic Automatic
// Type Conversion.
```

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 are:

Assignment Operators	
Operator	Description
=	Simple assignment
+=	Add and assign

Assignment Operators	
Operator	Description
<code>-=</code>	Subtract and assign
<code>*=</code>	Multiply and assign
<code>/=</code>	Divide and assign
<code>%=</code>	Modulo and assign
<code><<=</code>	Shift bits left and assign
<code>>>=</code>	Shift bits right and assign

The numerical operators available are:

Numerical Operators	
Operator	Description
<code>+</code>	Addition
<code>-</code>	Subtraction
<code>*</code>	Multiplication
<code>/</code>	Division
<code>%</code>	Modulo (remainder)
<code><<</code>	Shift bits left
<code>>></code>	Shift bits right
<code>&</code>	Bitwise AND
<code> </code>	Bitwise OR
<code>^</code>	Bitwise Exclusive OR
<code>~</code>	Bitwise Complement

Logical expressions evaluate to a one or zero, TRUE or FALSE, respectively.
Logical operators available are:

Logical Operators	
Operator	Description
==	Equivalence
!=	Non-equivalence
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
&&	AND
	OR
!	Negation

Special Operator Rules for Strings

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 this call in your macro:

```
MacroIgcase (FALSE) ;
```

- To query the current state use:

```
MacroIgcase (-1) or MacroIgcase ()
```

Special Operator Rules for Arrays

The following operator rules apply to API Macro arrays.

For the + and += operators these steps are used:

1. Convert the right operand to the type of the left.
2. Create a larger array by appending right operand elements to left operand.

For the logical comparison operators (>, <, >=, <=, !=, ==) these steps are used:

1. Convert the operand with the more complex type to the one with simpler type. (An array is more complex than an int or string, a string is more complex than an int.)
2. Then if both operands are arrays:
 - a. Compare each element with the same index, from first to last, until the elements do not match or no elements remain in one or both arrays.
 - b. If all elements match but one array has more elements, it is taken as having the greater value.

Determine the Size of an Array or String within a Macro

Use the API Macro built-in function **dim(<arr_name>)** to find the length of an array or string.

Other Operators

Refer to the following descriptions of other available operators.

Other API Macro Operators	
Operator	Description
,	<p><u>Comma operator.</u> This operator is used to:</p> <ul style="list-style-type: none">■ Separate parameters in the calls to other macros or functions (most commonly).■ Separate 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. The value taken for a sequence of comma-delimited sub-expressions is that of the right-most sub-expression.
?:	<p><u>Conditional operator.</u> 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>

Other API Macro Operators	
Operator	Description
++	<p><u>Pre-increment or post-increment operator.</u> 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><u>Pre-decrement or post-decrement operator.</u> 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.

Operators (from high to low precedence)	Associativity
() []	none
+ - ~ ! ++ -- (all unary)	R to L
* / %	L to R
<< >>	L to R
+ -	L to R
< > <= >=	L to R

Operators (from high to low precedence)	Associativity
<code>== !=</code>	L to R
<code>&</code>	L to R
<code>^</code>	L to R
<code> </code>	L to R
<code>&&</code>	L to R
<code> </code>	L to R
<code>? :</code> (trinary operator)	L to R
<code>= *= /= %= += -= <<= >>= &= =</code> <code>^=</code>	R to L
<code>,</code> (comma operator)	L to R

Parentheses in Expressions

Left and right parentheses, '(' and ')', may be used to group expression parts to set evaluation precedence.

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 CodeWright API functions and macros, take the form:

```
<macro_name> (<parameter1>,<parameter2>,...<parameterN>)
```

The following rules apply:

- Spaces and tabs are allowed after the macro name and between parameters.
- Commas must separate parameters.
- 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 needn't contain anything more than just 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-else' statement

```
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 <statement> 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 containing 'switch', 'while', 'do-while' or 'for' statement.

- '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 <incr_expression> is evaluated next.

Run-time Error Handling

API macros protect against several kinds of run-time errors:

Ctrl-break stop

If you've written a locked loop in your macro, use the  [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**.

Illegal Array Access

An attempted access of an illegal array index will generate a runtime error. An illegal index is one that is negative or greater than or equal to the size of the array.

CodeWright Event Handling

Three special macros have been added to CWRIGHT.MAC to facilitate event handling by API macros:

- **_Init_API_Macro** -- This macro is automatically called when CodeWright 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

In C, but not in API macros:

- Types 'char', 'short', 'float', 'double', 'unsigned'
- 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
- Arrays can have multiple dimensions
- Arrays can have a defined size for a parameter variable
- Pointers can be used to access array elements

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
- Automatic type conversion from arrays to expected types

- Automatic memory management for arrays
- Array types can be assigned to arrays (the resulting array sized to match)
- Array initializers can be used in expressions (not just for assignment in an array declaration)

String functions

Since library functions are not available in API Macros, a number of string functions are made available through the CodeWright 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.
- **UINT StringChar(LPWSTR str, int idx);**
Gets a character at idx from the string.
- **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);**
Transforms the supplied filename as indicated by the accompanying format controls and transformation patterns.
- **BOOL StrFileMatch(LPSTR fpattern, LPSTR fname, BOOL igcase);**
Determine if a filename is matched by a pattern.

Making and Modifying CodeWright DLLs

Loading DLLs, or Add-Ons, interactively from CodeWright's **Customize | Libraries** dialog can add functionality to CodeWright. CodeWright provides resources to create new DLLs and resources to modify existing fundamental DLLs. These resources and reasons for using them are described next.

Consider modifying or creating a CodeWright DLL only if the following is true:

- CodeWright 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 CodeWright DLLs is provided with a FULL installation of CodeWright. A full CodeWright installation also installs a **SAMPLE** subdirectory in the CodeWright 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 CodeWright through DLLs:

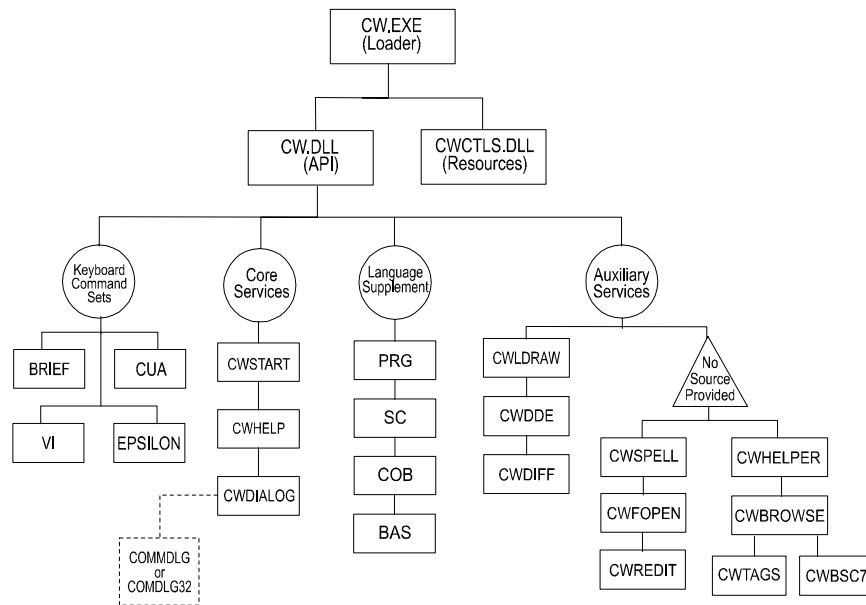
- It provides direction to the source code to be changed or added to.
- It provides some assistance for making the changes or additions.
- It provides tips on how to compile and make use of the changes or additions that have been made.

The first step to changing CodeWright is to understand its anatomy. The primary executable, CW32.EXE contains the CodeWright Application Programming Interface (API). These are the functions and capabilities that are basic to CodeWright's operation. Although you cannot rewrite the functions in the API, you can replace them, in a limited fashion. We will discuss function replacement later. The point is, 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:

- Core services.
- Supplemental language support.
- Keyboard command sets.
- Auxiliary services.

The following illustration depicts the organization of CodeWright's software system. The auxiliary and language supplements are not complete. CodeWright offers more in these areas than would fit on a page. However, the picture gives an accurate idea of CodeWright's architecture.



Core Services

The core services are those that CodeWright always expects to have available. Some of these services are so essential that CodeWright is unable to properly initialize without them. For example, you could use CodeWright 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 CodeWright 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 utilize them in your own source code you will need to include the header file CWSTART.H, which is in CodeWright's CWSTART subdirectory.

The file descriptions below will help you locate the functions provided in the various source files for CWSTART.

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 (e.g., <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).
CPPC	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).

Language Features	
File	Description
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).

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.
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 CWRIGHTMNU.
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 Word Wrap and related features.
DLGUTIL.C	Contains functions for enhancing CodeWright dialogs.
NAMEMARK.C	Contains functions for naming CodeWright's bookmarks.

Utilities	
File	Description
OUTLINE.C	Contains functions for CodeWright's Outline Symbols feature.
PROJECT.C	Contains certain functions relating to CodeWright projects.

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.

Macros	
File	Description
BRACE.C	The source code file for the CodeWright brace matching features, including the functions Brace , BraceMatch and BraceMatchNext .
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.
PROMPTC	This source file contains the prompting functions and supporting functions that use the prompt history facility.
REPEATC	Contains the source code for the function Repeat , which repeats a command a specified number of times.
SLIDE.C	Contains the functions for moving a block of text in or out.
TABSETC	Contains the functions supporting setting tabs, including extension specific settings.

Macros	
File	Description
TAGS.C	The source file for the functions that support the CTags features.
EXECMAC.C	Contains source code for CodeWright's API Macros.

CWDIALOG DLL

Many of CodeWright's 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 with CodeWright.

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 CodeWright's CWDIALOG subdirectory.

CWHELP DLL

The CWHELP.DLL contains the functions that access the various Help libraries used by CodeWright. These include the contextual help for the editor, the CodeWright 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 **cwhelp** from your C source code, you will need to include the file CWHELPH in your file. This header file is located in CodeWright's CWHELP subdirectory.

Keyboard Command Sets

CodeWright is supplied with four keyboard command sets, also called keymaps. They are: CUA (Common User Access), BRIEF emulation, Epsilon emulation, and 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 CodeWright's keymaps, refer to the topic *Using Keymaps* in the chapter *Custom 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. Here is a description of the language support DLLs provided:

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).

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 CodeWright configuration file (CWRIGHT.INI).

Here 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 CodeWright. 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 CodeWright's 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.

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 OEMfixed, 8514oem, terminal, or similar font. If not, these lines will be displayed as international characters. It is not loaded automatically.
CWDDE.DLL	This DLL contains the functions that turn CodeWright into a DDE server. Other applications can then operate CodeWright 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 CodeWright's 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 CodeWright 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 CodeWright installation. You will find it in the SAMPLE subdirectory in the CodeWright home directory. You can put some flesh on this skeleton, or you may use it as an example of what elements go in to a CodeWright DLL.

The source code for `SAMPLE.DLL` is a skeleton of comments and a function stub or two. The comments show you where to insert additions or changes that you want to make to CodeWright. These insertions might include:

- 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 CodeWright 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 CodeWright. This section will help you understand what CodeWright requires and why. Let's take apart a CodeWright DLL and see what is typically there.

The `_init` Function

Whenever CodeWright 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 CodeWright 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 CodeWright.

Exporting Functions

The functions in a DLL, and the parameters for each, must be registered with CodeWright before you can make full use of them. If you don't register them with CodeWright, 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 required unless you tell it. The **LibExport** function does this.

Note: 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 CodeWright's 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 the best way to go, 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 CodeWright 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 CodeWright. 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 CodeWright, 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 4.0 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 NT 4.0, Millennium, and 2000 are case-sensitive in function names, whereas Windows 9x is not.

Creating New Keymap Command Sets

Keymaps have additional requirements over and above those of a CodeWright DLL. This section will walk you through the basic outline of a keymap.

Keymap `_init` Function

Like the `_init` function for any other CodeWright DLL, the Keymap's `_init` function typically contains a series of **LibExport** calls that make functions available to CodeWright. 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 things you will want to do 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 doesn't create a new window for each new file that is loaded. If the users of your keymap will expect to have vertical and horizontal scroll bars on their edit windows, now is the time to make that happen.

There are several functions that will assist you in setting the options you require. They are **SysSetFlags**, **SysSetDefault**, and **SrchSetFlags**:

- Use **SysSetFlags** to set options you might otherwise set from various CodeWright 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 -- and we mean *really* empty. The only keys or mouse actions that will work are those that CodeWright lets Windows process. This includes menus, , ,  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 CodeWright 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 them 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 "short cut" keys or accelerators on the menu. You will rely on the **MenuAddKeyString** function to do this.

Recompiling a DLL

Part of the CodeWright 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 CodeWright 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 CodeWright 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 between Microsoft or Borland when installing CodeWright.

As mentioned, once the makefiles are installed, they can be found in the same directory as the source code for the DLL that is being built. Microsoft makefiles have the same root name as the DLL they make, with no extension. Borland makefiles use the same root name as the DLL, but their extension is .MAK. If you did not choose a *Full* CodeWright installation, you would not have been offered the choice of Microsoft or Borland source code, and they would not be installed.

The makefiles that CodeWright supplies are specific to one compiler or another because of the compile and link command lines that they employ. Adapting them to a compiler other than Borland or Microsoft should be a fairly straightforward task.

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 CodeWright 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 CodeWright 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 CodeWright DLLs are largely the same as you would use for compiling any Microsoft Windows DLL. CodeWright DLLs must be compiled using Large Memory Model. If you specify your include directories on the command line, be sure to include the CodeWright 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 CodeWright'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 make reference to 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

The command line for invoking the linker is defined in the CWMAKE.INC file that is included at compile time. CWMAKE.INC is stored in the INCLUDE directory of the CodeWright installation directory.

The libraries used in this link command usually include LIBW.LIB, OLDNAMES, LDLLCEW.LIB and CWRIGHT.LIB:

- 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 runtime.
- 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 CodeWright kernel that support the CodeWright API.

You may need to link in other import libraries at times, for example, if you are making direct calls to routines in CWSTARTD.LL, 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 that is included at compile time 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. Borland supplies IMPORT.LIB to you for the Windows API and CWRIGHT.LIB comes from Starbase for the CodeWright 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.

Your DLL will need all of the same parts described in the preceding section *Dissecting a CodeWright DLL*, 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 CodeWright to load them when needed.

The following examples assume that you have put your DLL in CodeWright's home directory, or in a directory that your CWLIB environment variable points to.

Load Functions at Startup

Loading functions at startup may slow down the initial loading of CodeWright, but if you have a fast CPU and hard disk, you probably won't notice much difference in daily use. This is set automatically when the library is loaded in CodeWright's **Customize|Libraries** dialog. Loading the library in this dialog adds the following line to the [LibPreLoad] section of CodeWright's configuration file (CWRIGHT.INI).

```
LibPreLoad=myDLL
```

Load DLL as Needed

The second option, loading the DLL only as needed, can cause a short pause when the function is first called. It 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 the chapter *Configuration Files and Command Line Flags*, in this manual.

17- UNIX

At this time, CodeWright only runs on the Intel platform under Windows 95, Windows 98, Windows 2000, Windows Millennium, and Windows NT 4.0. A version that runs under the UNIX environment is not offered. CodeWright does, however, have some options that make it possible to edit UNIX files within the program. Before the files can be edited however, there must be a way to load them from the UNIX system into the Windows system. Some options for loading UNIX files are:

- Via NFS (Network File Systems) networking. CodeWright does not provide the tools for networking UNIX and Windows systems via NFS. They must be obtained elsewhere.
- Via FTP (File Transfer Protocol). CodeWright provides a built in FTP program for transferring files via FTP. (For more information on CodeWright's FTP program, see the chapter on *File Loading, Backup and FTP*)

Once the file-loading method has been established, there are some things that can be done, and things to be aware of, when using CodeWright to edit UNIX files. The following topics cover the following issues:

- Maintaining UNIX End of Line characters.
- Maintaining filename case and file securities between UNIX and Windows Environments.

End-of-Line (EOL) Characters

The primary difference between files that reside on Windows systems and files that reside on UNIX systems is the characters that make up the ends of lines. For this reason, CodeWright has an option that changes the carriage return key so that it only inserts UNIX end of line characters in documents. The next section describes how to use this option.


Make UNIX EOL Characters the Default

DOS end of lines are composed of a carriage-return and line-feed, and are the initial default. UNIX end of lines are line-feed characters only. To enable UNIX end of lines as the default, complete the following steps:

1. Select the **Customize | Language | Options** dialog.
2. Highlight all applicable file type extensions.
3. Check the **UNIX EOL** checkbox.
4. 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

To ensure that the correct end of line characters are being inserted for the file being edited, CodeWright offers an **Auto-sense File Type EOL** option that automatically detects and turns on the correct end of lines when a file is being loaded. When selected, CodeWright 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  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 | General** dialog.
2. In the **File Loading/Reloading** group, check the box **Auto-sense File Type EOL**.

Change EOL Characters in the Source File

You can easily change existing EOL characters in your source file with a simple Search/Replace operation. Do this by searching for one type of EOL character and replacing it with another.

The Hexadecimal Values for End of Line Characters are:

- 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 the chapter on *Search and Replace and Navigational Tools* for more information.

Example:

If you wanted to change your DOS EOL's to UNIX EOL's:

1. Check **UNIX EOL** for the appropriate file type in **Customize | Language | Options**.
2. Go to the **Search | Replace**.
3. Make sure you have the **Regular Expression** box checked.
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 your DOS EOL's to Macintosh EOL's:

1. Go to the **Search | Replace**.
2. Make sure you have the **Regular Expression** box checked.
3. Search for **\n** and Replace with **\r**.

Macro for Automating UNIX EOL Conversion

CodeWright offers an API macro that will do the UNIX EOL conversion automatically. API macros are described in the chapter on *Extend CodeWright*. 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**, and then click **Run API Macro <UNIXEOLs>** on the **Tools** menu.
4. The macro will search the current document for all EOLs, and replace them with UNIX EOLs.

Compiling UNIX Programs from CodeWright

CodeWright executes compile commands by shelling to DOS and executing the command, as though it was being run at a DOS prompt. The output of the compile/build is then captured and presented in CodeWright's Output Window.

If you compile UNIX programs, you will need to do this outside of CodeWright, or find a way to access your UNIX system from within CodeWright.

- One possible solution is to use a Telnet implementation to issue commands automatically from a file.

Example: Set up a file with necessary commands to do a build. Then use the **Build** tool in CodeWright to launch a Telnet session to process the file. CodeWright will parse the resulting output and step 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 CodeWright via an NFS mounted disk drive.

Although Starbase 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.

Preserving Filename Case and File Securities between UNIX and Windows Environments

Filename case can be lost when saving files to UNIX systems from CodeWright. To have UNIX system case-sensitivity in CodeWright, check **Allow Case Sensitive Filenames** in **Customize | Environment | General**.

The loss of filename securities can also be an issue using CodeWright's traditional file-saving method. To preserve file securities on UNIX files, use CodeWright's File Rewrite Save Method to save files (also on the **General** tab).

To select these options, complete the following steps:

1. Select **Customize | Environment | General** dialog.
2. Mark **Use file rewrite save method**, and **Allow Case Sensitive Filenames**.

Chapter 18

18- Configuration Files & Command Line Parameters

This chapter gives an overview of the main configuration files used by CodeWright. In most cases it will not be necessary to edit CodeWright configuration files, but in the event that this becomes necessary, the information in this chapter gives an idea of what those files do. This chapter also talks about command line options that can be used from the CodeWright shortcut or other command lines to control the way CodeWright loads.

Configuration and State

The following sections discuss the location and contents of configuration and state files within CodeWright.

Location of the Configuration File

CodeWright's configuration data is kept in a separate file. During initial installation, the configuration file CWRIGHT.INI is placed in the same directory as your CodeWright executable file. You may move it to another location, if you like, or you may have several configuration files in different work areas. You may even specify another name for the file.

If your CodeWright executables are installed on a network, you will almost certainly want to place your configuration file somewhere else. By placing your configuration file in a private or local directory, you ensure that you will not be using or changing someone else's configuration.

CodeWright looks in several places to determine what configuration file to use. When it finds a place where you have indicated the location of the configuration file, or the file itself, it looks no further and proceeds to read the file.

Here are the places that CodeWright looks, in the order of searching:

1. It looks on the command line to see if you specified the /c parameter.
2. It looks in your environment for a variable named CWINI.
3. It looks in your working directory for a file named CWRIGHT.INI.
4. It looks in the directory containing CW.EXE or CW32.EXE, for the file CWRIGHT.INI.
5. It looks in the Windows directory for the file CWRIGHT.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.

Example: The string may take this form when specifying a directory:

```
D:\SOURCE\PROJECT1
```

It may also take this form when specifying a file:

```
D:\SOURCE\PROJECTS\CENCOM.CFG
```

If CodeWright does not find a configuration file at all, it will create one in the working directory when it needs one. CodeWright has no problem running without a configuration file, but it does notify you if one could not be found.

Introduction to Configuration and State

To properly understand CodeWright's projects and workspaces, you need to have some knowledge of how CodeWright maintains its configuration and how it preserves its state between sessions. This will help you intelligently choose which information to keep in which file. In addition, you may find it expedient to modify CodeWright's configuration file directly. This section will give you the information to do these things.

Configuration File

The configuration file normally contains information about the way you have set up CodeWright. This information may be put there during installation, through modifying the settings in dialog boxes, or by directly editing the configuration file. In the latter case, the configuration file might contain almost any CodeWright commands. The configuration file is named CWRIGHT.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 CodeWright configuration file include: [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:

- ✓ Keywords are actually the names of functions in the CodeWright API.
- ✓ The string assigned to the keyword contains the parameters required by that function.
- ✓ The assigned string may contain white space.
- ✓ Parameters may 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 you had open the last time you exited CodeWright. It also contains the more transient information about your CodeWright 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 CWRIGHT.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. Note that the project file overrides any information duplicated in the configuration file.

As you change settings in the various dialog boxes, you are automatically updating your configuration. You may select which file the dialogs use to update configuration information: the configuration file or the project file. You may further select which categories of information are kept in which file. For more information on how to store configuration information with your project see *Storing Configuration Options with a Project* in the chapter *Projects, Project Spaces and Workspaces*.

The main question you need to ask yourself when deciding where to store configuration settings is "Do I want these settings to travel with the project, or have them apply to multiple projects?"

- To have settings travel with a project, store them in the project file.
- To apply settings to multiple projects, store them in the configuration file.

Example File

Here 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


Here 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 will cause **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  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 CodeWright controls the contents of this section. You are not encouraged to 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 CodeWright, 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 CodeWright. In most cases, it is not necessary to directly edit CWRIGHT.INI.

Processing At Startup

When CodeWright is launched, the command line is processed before the configuration file or the state file. This gives the command line the opportunity to specify the location of these files. It also means there is some potential for the configuration file to reverse the effects of a parameter specified on the command line.

Order of Processing

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.

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.

Descriptions of Sections

Each section heading represents a logical grouping of functions, as shown in the following list:

Heading	Functions
Colors	Functions to set the colors for text, messages and so on. This is done with the SysSetDefault function.
Compiler	Used primarily by CodeWright for saving associations between file extensions and compilers.
DefaultKeymap	The function that sets the initial key command set, usually BRIEF or CUA. For example, the statement <code>DefaultKeymap=BRIEF</code> will cause CodeWright to load BRIEFDLL and call its keymap function, which it assumes to be the same as the filename root -- BRIEF .
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 CodeWright for saving the parameters used by the File Find and File Grep 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 CodeWright for defining print margins and headers.
Toolbar	Defines modifications to the default toolbar and sidebar configuration after using Toolbar Configuration dialog.
Template	Contains user-defined Language Template definitions.
VersionControl	For defining command lines for Checkin , Checkout and several other related commands. The section is duplicated for each version control system defined, so that you may have separate commands for each.

You may place any CodeWright function under any of the section headings, but grouping functions logically under section headings will help 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 top of them.

User-Defined Sections

You may define other section headings and place other editor commands under those sections. These commands will not be processed automatically at startup, however.

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 CWRIGHT.INI. A section can subsequently be assigned to a keystroke in CodeWright.

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 CodeWright 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 CodeWright's **Customize | Keyboards** dialog. For more information about assigning keys through CodeWright's interface, see the chapter on *Custom 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.

Together, these two sections give a functionality similar to that of Microsoft's PWB.

Scrap Buffers

This next example makes good use of CodeWright's multiple scrap buffers. For it to be useful, you must first define more than one scrap buffer. To do this, enter a value of 2 or higher in the **Number of Scrap Buffers** field on the **Customize|Environment|Clipboard** dialog.

Use these 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 scrap buffer** option in the **Customize|Environment|Clipboard** dialog.

Relating Checkboxes to Functions

There is a **System Options** checkbox in the **Directories** tab of the **Project| Properties** dialog, and several more checkboxes 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 checkboxes to the contents of your configuration file or project file:

Checkbox 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

Checkbox Option	Related Configuration Function
Compiler Settings	BrowseSetFile CompilerAddBuild CompilerAssign CompilerNewExt TagSetFile CompilerAdd
Language Options	ExtColors ExtColorsAssoc ExtIndentEnable ExtIndentEnableAssoc
Version Control Setup	CheckInSetCmd CheckOutSetCmd
Filename Filters	FilterAdd FilterDeleteList
Clipboard/Scrap Options	ClipboardEnableSepStr ClipboardEnableTermStr ClipboardSetSepStr ClipboardSetTermStr ScrapSetCount

State File

Another important CodeWright configuration file is the state file. The state file is stored by default in the CodeWright home directory and is called CWRIGHT.PST. CWRIGHT.PST records CodeWright's condition at the time of exit. Storing state information is turned on by default, but can be turned off in the **Customize | Environment | State** dialog.

Location of the State File

You may specify the location and name of your state file in your CodeWright 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, CodeWright 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 CWRIGHT.INI it will look for CWRIGHT.PST.

Once again, if no location is dictated for the state file, and no existing file is found, the file is created in the working directory. CodeWright can operate just fine without a state file. In fact, you may turn off the saving of state information, and CodeWright 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 CWRIGHT.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:\STARBASE
```

- On the command line:

```
CW /sD:\STARBASE\CWRIGHT.PST
```

- In the CWRIGHT.INI file:

```
StateSetFilename=H:\HOME\ERICJ
```

Contents of the State File

The following are among the data kept in the 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

Category	Types of Data in State File
Prompt Histories	Search and Replace Command Key Open file
Miscellaneous	Search flags CodeWright's Window frame coordinates and extents

Command Line Parameters

In the previous sections on configuration files some command line parameters were mentioned that could optionally be used with CodeWright's command line to specify certain conditions to apply when CodeWright is launched. This section describes those and other CodeWright command line parameters.

CodeWright supports a number of command line parameters. You may add these parameters 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 CodeWright shortcuts or program items, the command line may be used to have each operate differently.

To edit the properties of a CodeWright shortcut, do the following:

1. Right-click on the CodeWright's shortcut (wherever the shortcut appears).
2. Select **Properties** from the popup menu.
3. Click on 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 tell CodeWright to use specific settings when being launched.

CodeWright allows 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 configuration and state file locations.
- It allows configuration and state files to override command line specifications. This is not normally a problem, but possible conflicts are noted in the descriptions below.

Filename

You may specify any number of files to edit on the command line, limited only by the command line length. You may use wildcards 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 you wish to allow the configuration file to be read before or after each parameter that you specify. 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, wherever applicable.

There are upper case and lower case versions of each of the command line parameters.

- 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, CodeWright 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 the table on the next few pages are depicted as preceded by a minus or dash character (-). While a slash will also work in most instances, these can be mistaken for filename paths and should be avoided.

Sample Command Line Parameters		
Type	Format	Description
Configuration Location Parameter	<code>-C<configLoc></code> <code>CW32 -c</code> <code>c:\source\proj1</code>	<p>Allows you to specify the directory or file in which configuration information is to be found. The <code><configLoc></code> argument names that directory or file.</p> <ul style="list-style-type: none"> ■ If <code><configLoc></code> names only a directory, CodeWright looks for a file named <code>CWRIGHT.INI</code> in that directory from which configuration is read. ■ If <code><configloc></code> names a complete path, including filename, CodeWright will attempt to read configuration information from that file. <p>For additional location information, refer to the topic <i>Location of the Configuration File</i>, in this chapter.</p>
Variation of Configuration Location Parameter	<code>-C-</code> <code>CW32 -c-</code>	Instructs CodeWright not to read a configuration file.

Sample Command Line Parameters		
Type	Format	Description
Go to Line Number	-G <lineNumber> CW32 -g215	<p>Tells CodeWright 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.</p> <p>This parameter will be overridden if you attempt to position the cursor in a file loaded as part of the state restoration.</p>
Heap Allocation Parameter	-heapalloc CW32 -heapalloc	<p>Instructs CodeWright 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.</p> <p>✓ This option is only available on the 32-bit version of CodeWright.</p>
Keymap Parameter	-K <keymap> CW32 -k mycua	<p>Names a default keymap to be used by CodeWright. If this parameter is used, the [DefaultKeymap] section of the CodeWright configuration file (CWRIGHT.INI) will not be read.</p> <p>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.</p>

Sample Command Line Parameters		
Type	Format	Description
Library Parameter	-L <library> CW32 -Lmyutils	<p>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 CodeWright to start up.</p> <p>The <library> argument is the name of the library to be loaded. If this argument does not name a path to the DLL, CodeWright will look in the current directory, the Windows directory and on the PATH, in an attempt to locate the DLL.</p>
Multi-Instance Parameter	-M CW32 -M	<p>Allows you to run more than one instance of CodeWright at a time.</p> <p>By default, multiple instances are disallowed. Your operating system must be Windows 95, 98, ME, 2000 or Windows NT 4.0 for this option to work.</p>
No Files Parameter	-N CW32 -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 CodeWright's opening splash screen. This can save a small amount of loading time.

Sample Command Line Parameters		
Type	Format	Description
Paragraph Parameter	<pre>-P <section> CW32 -p "Windows driver"</pre>	<p>Tells CodeWright to read the named section of the configuration file. The section will be read after all of the standard portions of the configuration file have been read, to give it priority over the other contents of the configuration file.</p> <p>The <i><section></i> 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.</p>
Limit CodeWright CPU usage on multi-processor machines	<pre>-Processor = d (d is a decimal number bit mask for the allowed CPUs used by CodeWright on startup)</pre>	<p>This argument can be useful for avoiding thread-related problems that appear only on multi-processor machines. It can also limit CodeWright's CPU usage, allowing more for other processes.</p> <p>Examples:</p> <ul style="list-style-type: none"> ■ '-Processor=1' limits CodeWright to CPU 0. ■ '-Processor=2' limits CodeWright to CPU 1. ■ '-Processor=3' limits CodeWright to CPU 0 and CPU 1.

Sample Command Line Parameters		
Type	Format	Description
State Location Parameter	-S <stateLoc> CW32 -s h:\home\ericj	<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, CodeWright looks for a file named CWRIGHT.PST in that directory, from which state information is read. If <stateLoc> names a complete path, including filename, CodeWright will attempt to read state information from that file.</p> <p>When this parameter does not appear on the command line, CodeWright 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 CWRIGHT.PST in the Working Directory, ■ in CWRIGHT.PST in the directory in which the CW.EXE or CW32.EXE file is located, or ■ in a location named in CWRIGHT.INI.

Sample Command Line Parameters		
Type	Format	Description
Variation of State Location Parameter	-S- CW32 -s-	Instructs CodeWright not to read a state file. State information, however, is not automatically reinitialized, and may be used subsequently.
Execute Function Parameter	-X <i><function></i>	Names a function to be invoked as part of CodeWright 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 <i><function></i> 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 is a file that may contain any valid command line parameters, including additional command files. CodeWright 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.

Appendix A

A- TagsWNN Utility

TAGSWnn (TAGSW16 or TAGSW32, depending on your platform) is the Tags program supplied with CodeWright 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 wish 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 CodeWright.

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 a plus sign (+), a comma (,), a semicolon (;), 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 (i.e. '+', ',', ';' or whitespace) to differentiate between the beginning of a comment and a filename character (since '#' is a valid element of a filename).

`-x {EXCLUDEFILE | @LISTFILE}`

Exclude the files specified by EXCLUDEFILE or exclude all files listed in LISTFILE using the same syntax described above.

`-t TAGFILE`

Add new generated tags to TAGFILE. This file may or may not exist. All tags from TAGFILE that were derived from files currently being parsed will be removed during the merge phase. This tagfile is assumed to be in one of this utility's 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

`-p COMPFILE`

Convert output tag file (specified by -t) into a compiled file suitable for use with CodeWright. The output flag (-t) and the CodeWright output format flags (-oc) must be specified for this flag, otherwise it is ignored.

`-l LOGFILE`

Output all activity to LOGFILE. The log file will be created in a LISTFILE format (i.e. 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 (\geq 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 (i.e. database formats, etc.).

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 CodeWright 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 CodeWright 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 (-afdlmsu). 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 LISTFILES) 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, but does well for frail human memory). 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", etc.

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 (-cdmstekuvcpxi). 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 LISTFILES) 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

(#define token() statement)

This option specifies tagging of macros defined via use of the preprocessor #define directive.

s struct globals

(struct token {})

This option specifies tagging of structures defined via use of the "struct" keyword and implicitly defined within C++ syntax variations.

t typedef globals

(typedef declaration token, token, ...)

This option specifies tagging of identifiers defined via use of the "typedef" keyword.

e enum globals

(enum token {})

This option specifies tagging of enumerations defined via use of the "enum" keyword.

k enum konstants

(enum { token, token, token, ... }) Note the cute spelling of constants with a 'k' to justify the assignment of this letter. This option specifies tagging of enumeration constants within declared enumerations.

u union globals

(union token {})

This option specifies tagging of unions defined via use of the "union" keyword.

v global variable

(declaration token, token = {}, token, ...)

This option specifies tagging of global variable declarations.

c global class

(class token: {})

This option specifies tagging of class definitions specified via use of the "class" keyword.

f function definitions

(token() declaration {})

This option specifies tagging of function declarations.

p prototypes

(token();)

This option specifies tagging of prototypes.

x extern defines

(extern declaration)

(extern "C" declaration)

(extern "C" { declaration; declaration; ... })

This option will specify that tags which have the extern storage class are to be output. The x option is a modifier and will only be effective when other options are used (i.e. -cpx must be specified to obtain extern prototypes, -cx alone yields nothing). Note also that the -cx modifier has no effect for function, define and macro tags which are tagged only according to the f, d and m options respectively. This modifier may be placed anywhere within the options list.

i static declarations

(static declaration)

This option will specify that tags that have internal static storage class are to be output. The i option is a modifier and will only be effective when other options are used (i.e. -cvi must be specified to obtain static variable declarations, -ci alone yields nothing). Note also that the -ci modifier has no effect for define and macro tags that are tagged only according only to the d and m options respectively. This modifier may be placed anywhere within the options list.

-d

This flag specifies that all input listfiles and exclude response files should be deleted once parsed. This allows an automated list file generation that is cleaned up by the tags package. See description of LISTFILE below.

-j

This is the junk filter switch to allow the filtering of functions and declarations that are overloaded operators in C++. For example, if the junk filters are enabled then the declaration "inline myType operator+(MyType m1, MyType m2);" would not be tagged for "+" which is normally a standard C delimiter token and operator. The junk filter, if enabled, will filter all standard C delimiters from the output.

-q

This is the quiet switch and will suppress normal status output to stderr and program version information.

-r

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.

-n

This switch will suppress sorting of the tag output (often used in conjunction with GNU or Epsilon style tags).

-i

This switch specifies the use of a case sensitive sort (normally a case insensitive sort is used). Although the character 'i' is normally used for switching to a case insensitive behavior, it differs in this instance.

Index

Symbols

`${FTEE}` 142, 163, 166
% macros 72
%Q macro 146
.CHM Help Files 115
.HLP Help Files 114
.HTML Help Files for MSVC 114
.IVT Help Files 115
.MVB Help Files 116
_init 393
_init Function 395

A

Accessing CodeWright Functions
 from Perl Scripts 339
accessing errors 151
Accessing Perl functions 340
activestate 344
Adding Your Own Functions 394
Add-Ons 327
 Language DLLs 50
Add-Ons/DLLs/Libraries
 Additional Add Ons on the CD 24,
 42
 Loading 24, 42
Alternation and Grouping 221
ANSI
 CodeSense translation 96
Answer Wizard 17
API 39
API (C-like) Macros 363
API Assistant 67, 110
 Automation Tools 113
 checkboxes 111
 creating a database 113
 databases 112
 Modifying the Database 113
 using 110
API Assistant Database Editor 113

API Command 80, 340
API Command Prompt
 API Command Key/Prompt 40
 Command Completion 40
 Command Key/Prompt 40
 using 40
API Commands 360, 363
API Functions 328
 Command Key 328
 Expression Evaluator 330
 Return Values 328
API Macros 135, 327
 CodeWright Event Handling 382
 Comments 366
 Conditional and Iterative Control
 Structures 379
 Creating and Editing Macros 364
 Differences between API Macros
 and C 383
 Expressions 371
 Identifier Naming Rules 366
 Language Definition 365
 Literal Values 368
 Run-time Error Handling 382
 Statements and Statement Blocks
 379
 String Functions 384
 Variables 366
AppBasic 327, 344
 Environment 345
AppBasic Macro Language 344
 AppBasic macros 349
 AppBasic Window 360
 AppBasic Window Configuration
 360
 AppBasic Window Toolbar 360
AppBasic-related API Commands
359
 Choosing an Editor 345
 Choosing Toolbars 347
 Creating a Handler 349
 Creating a Macro 348
 Creating an Event Handler 355
 Debugging Macros 356
 Dialog Editor 347
 Exported Functions in CWBA-
 SIC.DLL 361

- Initialization Subroutine 351
- Load Macros Dialog 358
- Modal User Dialog 352
- Object and Proc Lists 350
- Object Browser 348
- Pop-up Menu 347
- Related API Commands 359
- Running the Macro 355
- Special Keybindings 346
- User Dialog Editor 352
- Window Configuration 360
- assemblies 161
- assembly language 391
- Assign Keys dialog 301
- Auto Sync Makefile/ Visual Studio
Workspace 131
- Auto-hide
 - Toolbars 277
- auto-list members
 - CodeSense 95
- Auto-parameter Info
 - CodeSense 96
- Auto-Save Settings 308
- Auxiliary Services 392

B

- back to previous document position
31
- Backup Files 312, 418
- Backups
 - Format Controls 313
 - Formatting the Backup Specifica-
tion 312
 - Global 308
 - Individual Files 311
 - Specific File Types 310
 - Transformation Patterns 314
 - Turning off 321
- beautify
 - format source 64
- Beautifying Code 265
- Bi-directional synchronization 199
- Binding Keystrokes to Functions or
Macros 303
- Block Alignment 84
- Bookmarks 134

- Global 247
- Local 247
- Bookmarks Window 249
- Borland C++ Builder File Synchroni-
zation 193
- Borland C++ File Synchronization
191
- Borland Link 399
- Brace 80
 - Align 82
 - automatic positioning 82
- Brace 80
- Brace Expansion 82
- BraceFindEx 81
- BraceMatch 81
- BraceMatchNext 81
- highlighting 81
- kissing 81
- locating 81
- locating square brackets 81
- matching 80
- matching parentheses 81
- parenthesis locating 81
- Brace Expansion 67, 82
- BraceFind 81
- BraceMatch 81
- BraceMatchNext 81
- break
 - a long search 214
- Break Points 356
 - AppBasic 357
- BRIEF 40, 297, 328
- BRIEF command set 295
- BRIEF Keymap 295
- Browse
 - C++ and Java 242
- Browse Object 348, 358
- Browser
 - Database 134
 - Filter Toggle Buttons 234
 - Inspect window 230
 - Jump to Code 232
 - Objects Window 229
 - Query Buttons 232
 - Selecting a Database 230
 - String Search 232
 - Support 229

- Toolbar 230, 231
- Traversing the Tree 230
- Tree window 230
- Window 230
- Browser tree
 - expand or collapse 230
- Browsers 247
- Browsing
 - using Microsoft .BSC files 229
- BSC file 230
- Bufsethexascii 329
- Build 151, 161, 166
- Build Command Line 163
- Build Tool 142
- build utilities 137
- builds 161
- Button File 135
- Button Link 134
- Button Links
 - Comment prefixes 250
 - Defining buttons 251
 - How it works 250
 - Insert Link 20
 - What you see 251
- Buttons 20, 250
 - action 20, 250
 - Adding and Changing Toolbar Buttons 280
 - Binding a Function to a Button 281
 - Information File 282

C

- C Language Operators 330
- C++
 - Browse 242
- Called By 234
- Calls To 234
- CARP.PM 343
- Case
 - Changing Upper, Lower 22
- Centering text 22
- Changing Existing Functions 394
- Character Classes 53, 59, 219
- Character Matching 219
- ChromaCoding 365, 387
 - Adding a New File Type 64

Index

- COBOL 103
 - color-timing 63
 - enable coloring 65
 - for embedded languages 63
 - language specific editing features 65
 - maximum number of comment-lines 63
 - Numbers 58
 - Strings 57
- ChromaCoding Lexer Settings
 - Default 52
 - Item Settings 56
 - List Settings 56
- ChromaCoding Lexer Settings Dialog
 - 51
 - Comments tab 56
 - Numbers tab 58
 - Strings tab 57
 - Words tab 54
- ChromaCoding Lexers 24, 51, 62
 - add keywords 54
 - add operators 54
 - Adding Comment Delimiters 56
 - adding keywords from a file 55
 - Adding Numbers 58
 - Delete keywords 55
 - Deleting Lexers 60
 - Digits field 59
 - Exclude Text from Coloring 53
 - Identifier Characters 52
 - Identifiers 52
 - Lexer names 52
 - Multi-line Comment 57
 - Single Line Comments 57
 - user-defined keywords 55
- ChromaCoding Lexers--numbers 59
- Class browser 242
- Clipboard 109
- Clipboard Viewer 109
- Closed Selections 297
- COBOL 104
 - Automatic Time/Date Stamp 105
 - Automatically Continue Strings 105
 - DLL 103
 - Resequenece Line Numbers 104
 - Toggle comment 105

- Validate Line Number Sequence 106
- COBOL ChromaCoding support 103
- COBOLLine Number Handling 105
- Code Beautification 265
- Code Composer
 - Bi-directional synchronization 199
 - synchronization 197
- Code Reformatting 64
- Code Snippet
 - execute 75
- Code Snippet Properties Dialog 78
- CodeFolio
 - Tab on Project Window 29
- CodeFolio Snippets 29, 67, 74
 - adding 77
 - Adding or Removing Directories 79
 - Code Snippet Properties Dialog 78
 - Creating a Snippet from Clipboard or Document Selection 77
 - Deleting or Renaming 78
 - directories 75
 - Editing 78
 - using 75
- CodeSense 24, 86
 - access definition 94, 95
 - ANSI translation 96
 - Auto-list Members 95
 - Auto-parameter Info 96
 - Auto-type information 95
 - Create library database 89
 - database files 92
 - delete library database 90
 - edit library database 89
 - Extended Functionality 96
 - from files that are open in CodeWright 88
 - Global Configuration 89
 - libraries 88
 - Main Features 94
 - Name Completion 94
 - parser priority 91
 - Symbol Lookups 97
 - Troubleshooting 98
 - Unicode translation 96
- CodeWright API 328
- CodeWright API functions 39, 334
 - use interactively 39
- CodeWright Event Handling 382
- CodeWright Libraries 24, 42, 51
- Collapse text 226
- Colors 45
 - changing colors 46
 - changing individual screen elements 46
 - changing whole background only 46
 - ChromaCoding Lexers 51
 - for embedded languages 63
 - Synchronize text background 46
- Columns
 - Formatting 22
 - Marking 297
 - Selection 297
- Combo Box History Lists 282
- Combo box lists 282
- Command Files 426
- Command Key 40, 80, 227, 328, 330, 419
 - API Command Prompt 40
 - BRIEF keymap 40
 - CUA keymap 40
 - examples 329
 - using 40
 - VI keymap 40
- command line length limits 146
- Command Line Parameters 4, 419
 - Command Files 426
 - Order of Processing 419
 - Samples 421
 - Types 419
- Command Options 143
- Comment
 - Comments 99
- Comments
 - API Macros 366
 - Comment Box 99
 - Comment Boxes 67
 - comments and comment boxes 64
 - inserting 22
- Common User Access 294
- Compact Mode 227
- Compile 161, 166
 - command line compilers 137

- compile a specified target 164
- Compile and Link Options 398
- compile tools 139, 162
- compile utilities 137
- compiled tags database (.ptg) 134
- Compiler 151, 162, 163
 - batch files 165
 - command line 163
 - command line length limits 146
 - directory structure 164
 - setting up 161
- Compiler commands 161, 162
- Compiler name 162
 - creating, deleting 162
- Conditional statements
 - API Macros 379
- CONFIG.PM 343
- Configuration
 - Project Files 409
- Configuration and State
 - Command Line Parameters 419
 - introduction to 408
- Configuration File 4, 421, 422, 424
 - Command Line Parameter 408
 - Contents and Purpose 408
 - Example 410
 - Location 407
 - Options (Dialogs and Functions) 416
 - Processing at Startup 412
 - User-Defined Sections 414
- Configuration Hierarchy 159
- configuration settings
 - reading from other files 136
- Configuration Wizards 17
- Control Structures
 - API Macros 379
- Copy 108, 299
- Core Services 387
- CPAN 344
- Creating a ChromaCoding Lexer 52
- Creating a Macro
 - API Macros 364
- Creating a Project 34, 125
- Creating and Editing Perl Scripts 335
- Creating Windows With A Mouse 300
- Ctags Support 235
- CUA 40, 294, 295, 297, 328
- CUA Keymap 295
- Cursor
 - Placement After Search 223
- Custom Error Parsers 149, 150
 - making 150
- Custom tools 25, 140, 161
 - creating 140
- Customization 331
- Customize
 - using your own DLL 400
- Customize Menu 23
 - Environment option 42
- Customize|Language 61, 62, 84, 85, 311, 313, 315
- Customize|Libraries Dialog 50
- Customizing the CodeWright Interface
 - Dockable Toolbars and Windows 275
 - Keymaps 294
 - Menus 283
 - Mouse Commands 296
 - Toolbars and Buttons 279
- cwbLoadFile 355, 359
- cwbShowProcDisplay 360, 362
- cwbShowToolbar 360, 362
- cwbUnloadFile 359
- CWConst 341
- CWD 206
- CWDDE.DLL 392
- CWDIALOG.DLL 390
- CWDIALOG.H 390, 394
- CWExec 342
- CWHELP.H 394
- CWLDRAW.DLL 392
- CWP.DLL 339, 343
- cwp.dll 334
- CWP.PM 334, 339, 343
- CWP.XS 343
- CWPERL.DLL 343
- CWPERLI.DLL 343
- CWPERLI.MNU 343
- CWPerlIO 342
- CWRIGHT.INI 355, 360, 391
- CWRIGHT.MAC 363
- CWRIGHT.PST 425

CWSTART.DLL 387, 391
CWSTART.H 387, 394

D

- d flag 340
- data types 348, 357
- Debug 161
- Debug Compile 151, 161
- Debugging using expression evaluation 330
- Debugging Your AppBasic Macro 356
- Default
 - Backup Specification 313
 - Keymap 422
- DefaultKeymap 400
- Delphi File Synchronization 189
- Delphi's API 112
- Differencing
 - Advantages 253
 - Analysis Dialog (Interleaved) 254
 - Difference Analysis Dialog (Side-by-Side) 256
 - Interleaved 254
 - Interleaved Document 255
 - Interleaved Options 254
 - Reference vs. Target File 255
 - Side-by-Side 256
 - Side-by-Side Difference Controls 261
 - Side-by-Side Difference Window 257
 - Side-by-Side Edit Mode 259
 - Side-by-Side Options 257
 - Side-by-Side View Only Mode 258
- directories
 - Edit Search Path 135
 - system files 133
- Displaying Return Values 328
- Dissecting a CodeWright DLL 393
- DLL Extensions 333
- DLLs 327
 - Compile and Link Options 398
 - CWDIALOG.DLL 390
 - CWHELP.DLL 390
 - CWSTART 387
 - Dynamic Link Libraries 42

- Installing your DLL 400
- Libraries and Add-ons 24
- Link Libraries 398
- Makefiles 397
- Making Add-ons 385
- Organization of DLLs within CodeWright 386
- Recompiling 397
- Source Code 331
- Supplemental Features 392
- Supplemental Language Support 391
- Typical Contents 393
- Using your Own Custom DLLs 400
- DLLs/Libraries/ Add-Ons
 - loading 42
- Dockable Toolbars 347
 - manually dock and undock 278
- Dockable Toolbars and Windows 275
- Document
 - Difference between windows and documents 26
 - move forward and back 31
- Document Menu 22
- Document/Window Manager Dialog 23
- Drag and Drop
 - Copy 299
 - File Loading 300
 - Move 299
 - Text 299
- Drag and drop
 - projects and project spaces 300
 - Visual Studio workspaces and projects 300
- DYNALoader.PM 343

E

- Edit Menu 20
- Edit Search Path 135
- Edit UserDialog 352
- Editing a Macro
 - API Macros 364
- Editing features 65
- Elision 227
- Email 8

- embedded language support 63
- environment 45
 - setting up the 164
- Environment dialog and options 42
- environment space 165
- Epsilon command set 295
- Error File 148
- Error Handling
 - API Macros 382
- error output 147, 151
 - accessing compiler, linker and assembler 149
- Error Parsers 149, 150
- Error Parsing
 - Supporting Files 389
- errors
 - accessing 147, 151, 152
- EvalExpression 341
- Evaluate Expression and Add Watch
 - AppBasic 357
- event handler 355
- EventHandler in AppBasic 355
- Example Configuration File Settings
 - AppBasic 361
- Exclusive Selection 296
- Execute 161
- Exiting and Macro return values
 - API Macros 379
- Expand / Collapse 301
- Expanding And Collapsing Text 301
- EXPORTER.PM 343
- Expression Evaluator 330
- Expressions
 - API Macros 371
- ExtAssignTemplate 68, 69
- Extend
 - using your own DLL 400
- Extending CodeWright
 - API Functions 328
 - Changing and Adding Functions 394
 - Core Services 387
 - Creating New Keymap Command Sets 395
 - Keyboard Command Sets 390
 - Macro Languages and DLL Source Code 327

- Macros, Macro Languages and DLLs 331
- Making DLL Add-ons 385
- Supplemental Language Support 391
- Extensibility 331
- Extension File 134
- External documents/ programs- accessing
 - Button Links 250
- ExtExpandTemplate 70
- ExtSetTemplateMacro 73

F

- F1 key
 - Configuring MSVC HTML help 115
- Fast Find 214
- Fax
 - Number 8
 - support via 8
- File Finding
 - File Find tab 27
- File Grep 208
- File Headers 67, 69, 76
- File Menu 19
- file type
 - defined 61
- File View Window
 - Icons 174
- FILE.TPL 70
- Filename Component Macros 144, 145
- Filename Transformation 312, 313, 314
- Filenames 420
- Files
 - Attributes - Open Tab 30
 - Backups for Individual Files 311
 - Backups for Specific File Types 310
 - delays when opening/closing 240
 - File Manager 29
 - File Open dialog functionality 29
 - Finding 19
 - Formatting the Backup Specification 312
 - Global Backups and Auto-Save 308
 - loading 135

- Loading and Validation 305
- Open tab -- Project Window 30
- opening 135
- Opening as Read-only 317
- Read-only Files for Specified File Types 316
- Read-only for Individual Files 316
- Reloading current file from disk 19
- Saving Large Files 324
- Sending via Email 19
- Show File List on File Menu 43
- Working with Large Files 319
- Files used by Perl for CodeWright 343
- File-Validation 4
- Fill Pattern 314
- Filter 245
- Filters 19, 25
 - File|Filters dialog 19
 - project filters 154
- Find Fast 214
- first error 152
- Flag Initialization 395
- Flags 420
- Fmatch 413
- Fold text 226
- Fonts 45
 - print fonts 46
 - screen fonts 46
- Format Controls 313
- Format Source 22, 64, 265
- Formatting Source Code
 - Language Dialog 266
- forward to next document position 31
- FTEE 166
- FTEE.EXE 163
- FTP
 - Put and Get 318
- Full Screen
 - Window menu 26
- FUNCT.TPL 70
- Function calls
 - API Macros 378
- Function Completion
 - CodeSense 86
- Function Headers 67, 69
- Functions
 - Changing and Adding 394

G

- Getting Started with API Macros 364
- Getting Started with Perl 334
- Gnu Tags 235
- Grep
 - Search and Replacement Edit Boxes 205
 - Selective Display 13

H

- Handlers 348, 350
- Help
 - accessing from other environments 113
 - Compiled Microsoft HTML (.CHM) Help Files 115
 - configuring 113
 - DLL 390
 - Index File Wizard 17
 - InfoViewer 116
 - Integrating with MSVC 5.0 (.IVT) help 115
 - MSVC 4.0 (.MVB) Help Files 116
 - MSVC 6.0 HTML help integration 114
 - Standard toolbar 31
- Help Menu 27
- Hex Editing 107, 108
- Hex Mode 107
- Hexadecimal values 107
 - searching 225
- Hidden Characters
 - making visible 46
- Hidden files
 - Open tab-- Project Window 30
- Hide
 - text 226
- History Of Responses 282
- hotspots on the status line 298
- HTML
 - Editing 100, 101
 - language support 100
 - popup menu 100

- toolbar 100
- turn on WYSIWYG 101
- Using MSVC 6.0 Help Files 114
- View 100
- WYSIWYG 101
- WYSIWYG editor/viewer 100

I

- Identifier Characters
 - First Field and Follow Field 53
- Identifiers 52
- if-else statement 379
- Importing names into Perl's namespace 339
- Inclusive or Exclusive Selection 296
- Incremental Searching 215
- Indent
 - Alignment 84
 - Prompted Slide In/Out 22
 - Seek Indentation 84
 - Slide In 22
 - Slide Out 22
 - Smart Indenting 84
- indentation styles 84
- Indented Alignment 85
- Indenting 83
- InfoView 113
- Insert
 - Braces 82
 - Button Links 20
 - Comment 99
 - File 20
 - Function and File Headers 69, 76
 - functions 74, 110
 - Line Numbers 22
 - Link 251
 - Literal (Hex, Decimal, ASCII) 20
- Insert/Edit User Dialog 352
- Installing Your DLL 400
- Interleaved Differencing 254
- Internet Mail 8
- Interpreters dialog 341
- Introduction 327
- Iteration Qualifiers 220
- Iterative statements
 - API Macros 381

Index

J

- Java 112
 - Browse 242
- jump to first error 151, 152

K

- Key Assignments 413, 422
- Key Command
 - Command Completion 40
- Keyboard
 - Assign Keys 24
- Keyboard Command Sets 390
- Keymap Function 395
- Keymaps 390
 - Binding Keystrokes to Functions or Macros 303
 - BRIEF Key Commands 295
 - Creating New Keymap Command Sets 395
 - CUA Key Commands 295
 - Customizing 294
 - Customizing Keybindings 301
 - Specific Assignments 396
 - Specific Keymap Assignments 301
- Keystroke Macros 20, 135
 - Editor 302
- Keystrokes
 - modifying 301
- keywords
 - from file 63
- KmapAssign 396

L

- Language
 - coloring 49
 - coloring embedded 62, 63
 - Customize Menu 24
- Language Definition
 - API Macros 365
- Language Dialog 68
 - ChromaCoding 62
 - Comments Tab 64

- DLL radio button 62
- Format Tab 64
- Lexer radio button 62
- Options Tab 61
- Symbols Tab 64
- Tabs/Indenting Tab 62
- Templates Tab 62
- Language DLL 50, 65
 - Creating a Language Support DLL 66
- Language Features
 - Supporting Files 387
- Language Indenting 387
- language support 49, 391
 - Adding a New File Type 64
 - additional language support 50
 - Alias or map non-supported languages to supported ones 65
 - ChromaCoding Lexers 51, 62
 - COBOL 103
 - Creating a ChromaCoding Lexer 52
 - Creating a Language Support DLL 66
 - default language support 49
 - enable coloring 65
 - Language DLLs 50, 65
 - language specific editing features 65
 - loading 50
 - setting configurations 60
 - user-defined keywords 55
- Large Files 319
 - Changing Block Size 320
 - Changing the Number of Swap Blocks 319
 - Disabling File Pre-loading 322
 - File Rewrite Save Method 325
 - Saving 324
 - Swap Blocks Allotting Memory to Swap Blocks 319
 - Turning off Backups 321
 - Turning Off Scroll Bars 321
- Left Justify 22
- Lexers 62
 - COBOL 104
 - Default Settings 52
- LibExport 339, 348, 393

- LibFunctionExec 74, 339, 348
- LibFunctionExistsWhere 333
- Libraries/ Add-Ons/DLLs
 - loading 24, 42
- Line
 - Numbering 22
 - Saver Alignment 85
 - Selections 297
- Link
 - insert 251
 - Insert Button Link 20
- Link Libraries 398
 - Borland Link 399
 - Microsoft Link 398
- Literal
 - Insert Hex, Decimal, or ASCII Literal 20
- load and execute a Perl script 338
- Load Macros Dialog
 - AppBasic 358
- loaded for execution 351
- Loading and running scripts 336
- Loading and Validating Files 305

M

- Macro File 135
- Macro Languages
 - API (C-like) Macros 363
 - AppBasic 344
 - Comparison of Supported Languages 332
 - Perl 334
- Macros
 - Binding Keystrokes to Functions or Macros 303
 - Creating a Macro in AppBasic 348
 - Debuggin in AppBasic 356
 - Defined 331
 - In templates 70
 - Keystroke Macro Editor 302
 - Loading a Perl Macro 337
 - Recording a Keystroke Macro 302
 - run from Button Links 251
 - Running AppBasic Macros 355
 - Supporting Files 389
 - Tools Menu options 25

- Using Macros vs. DLL Extensions 333
- MACROS subdirectory 335
- Mail
 - Sending files 19
- make utilities 137
- Makefile Parsers 131
- Makefiles 397, 398
 - adding files to CodeWright projects 129
 - Auto-Sync 131
 - reading into projects 131
 - Synchronize with CodeWright projects 131
- Making Changes and Additions 394
- Mark Database 134
- MarkBeginSel 341
- Members tab of Project Properties Dialog 36, 128
- Memory
 - Block Size 320
 - Pre-loading Files 322
 - Swap Blocks 319
- Menu Accelerators 396
- Menu items 284
 - adding 284
 - deleting 284
 - separator 284
- MenuAddKeyString 396
- Menus
 - adding 283
 - Adding a Menu Item 286
 - Adding External Operations 287
 - changing position 283
 - Changing the Functionality of a Menu Item 285
 - Customizing 283
 - Defining a Popup Menu 288
 - deleting 284
 - Document Menu 22
 - Edit Menu 20
 - Editing Menu Items and Submenus 284
 - Editing Top-Level Menus 283
 - file lists 284
 - File Menu 19
 - Project Menu 21
 - Search Menu 20
 - Sending Menu Commands to Synchronized Environments 201
 - Text Menu 22
 - Tools Menu 25
- Merging Files 262
 - Output 263
 - Removing Changes 264
- Metacharacters 219, 220, 221, 222, 223
- Microsoft Browser Database (.BSC) 134
- Microsoft Foundation Classes 112
- Microsoft Link 398
- Microsoft Visual Studio
 - Bi-directional synchronization 199
 - Drag and drop workspaces and projects 300
 - Synchronization 187
- Microsoft Visual Studio Workspace
 - Auto Synchronize 131
 - drag and drop to create project space 129
 - opening as project space 129
 - Parsing 131
 - Reading 131
- Modal User Dialog in AppBasic 352
- Modules Properties dialog 348, 349
- Mouse 297, 299, 300
 - Copying Text 300
 - Creating Windows With 300
 - Moving Text 299
- Mouse Commands 296
 - Block Selection 296, 297
 - Column Marking 297
 - Copy and Move 299
 - Creating Windows 300
 - Expand/Collapse Sections of Text 301
 - File Loading 300
 - scrolling speed 296
 - Selecting Lines 297
 - Selecting Words 298
 - Status Line Actions 298
 - Text Drag and Drop 299
- Move 299
- MsgNotify 349
- MSVC workspaces and projects

- Drag and drop 300
- MSVC++ File Synchronization 187
- Multiple Sources Search Dialog 205
- Multi-source search
 - Fast Find 214

N

- name confusion 333
- Navigate
 - move between documents 31
- Newline Character 69, 218
- Next Error 152
- Nmake 142
- No Command Shell 143
- note links 251
- NotePad 294

O

- O'Reilly Press 344
- Object and Proc Drop Down Lists 350
- Object Browser 348, 357, 358
- Objects tab 229
- Objects Window 242
 - Filter/Legend dialog 245
- OLE Automation 348
- One Document per Window 26
- Online Help
 - AppBasic 347
 - Perl 336
- Open projects and project spaces 129
- Open tab
 - Project Window 29
- option 25
- Outline Symbols 238
 - Auto-Expand Collapse 239
 - sorting 239
- Outline Window 229, 236, 238
- output
 - capturing 143, 147, 166
- Output Window 27, 151, 208, 229, 230, 334, 336, 348
 - AppBasic tab 344
 - Browse tab 28
 - Build tab 27, 151, 166

- ClipView tab 109
- Difference tab 28
- Perl tab 335
- Search tab 28
- Shell tab 28
- Symbols tab 28
- Override Pattern 314

P

- Parameter Checking 111
- Parameters 420
- parentheses
 - locating 81
 - matching 81
- Parentheses in Expressions
 - API Macros 378
- Pascal 391
- paste 108
- Pctags 235
- Perl 327, 334
 - books 344
- Perl Macro Language 334
 - Accessing CodeWright Functions 339
 - Accessing Perl Functions 340
 - already installed 334
 - Creating and Editing Scripts 335
 - Debug Mode 340
 - Files used within CodeWright 343
 - help 336
 - Importing Names into Perl's Namespace 339
 - Loading a Macro 337
 - Loading and Running Scripts 336
 - Other Resources 344
 - Perl Tab on Output Window 335
 - Pop-up Menu and Options 336
 - Special API Functions 341
 - Specifying the Intended Subroutine 341
 - Unloading a Perl Macro 340
- Perl options dialog 340
- Perl popup menu 342
- Perl Window 335
- Perl.HLP 336
- perl.org 344

- PERL_IN_ 342
- PERL5DB.PL 343
- PerlExec 338, 340
- PerlExecStr 338
- PerlExecSub 338
- PerlLoad 338
- PerlRun 338
- PerlUnload 340
- Phone Number 8
- PL extension 335
- Pop-up Menu 288
 - AppBasic 347
 - Supporting Functions 294
 - Version Control 175
- Pop-up Menu and Options
 - Fast Find 214
 - Perl 336
- Pre-loading Files 322
- Preprocessor coloring
 - Ignore Remainder of Line 56
- Preprocessor directives
 - selective display 227
- PRG.DLL 391
- print configurations 117
- Print Preview 118
- Printing 117
 - color 118
 - date, timestamps, page number, filename 119
 - header and footer macros 119
 - Headers and Footers 118
 - line numbers 118
 - multi copy 118
 - Paper Selection Override 117
 - Wrap Long Lines 118
 - wrapping long lines 118
- Processing At Startup 412
- Project and Output Windows 27
 - toggle 26
- Project Files 159
 - format 409
 - grouping 153
 - loading 156
 - Options 416
- Project Menu 21
- Project Properties Dialog 230
 - Directories tab 133, 134
 - Errors tab 147
 - Filter tab 153
 - Members tab 35, 127
 - Tools tab 137, 140
- Project Setup Checklist 154
- Project Space 122
- Project Spaces 121
 - creating 33, 122
 - Defined 122
 - Drag and drop 300
 - drag and drop 129
 - opening or loading 129
 - selecting or changing 155
- Project Tools 137
 - Build command 164
 - Command Options 143
 - Running in the Background 143
- Project Window 28, 132, 133, 156, 229
 - Bookmarks tab 29
 - CodeFolio tab 29
 - File View Icons 174
 - File View tab 28
 - Files tab 132
 - Objects tab 28, 229, 242
 - Open tab 29
 - Outline tab 28
 - refresh 133, 174
 - version control 132
 - version control operations 133
 - viewing version control status 133, 174
- Projects 121
 - add files 35, 127
 - adding files from external make-files/Workspaces 129
 - Associating Version Control Projects with 171
 - Auto-Update Symbols database 135
 - Command lines 154
 - Command Options 143
 - Compile commands 155
 - Compiling 155
 - Compute Document Symbols 135
 - creating 34, 125
 - Custom tools 140
 - defined 121
 - Directories 154

- Drag and drop 300
- drag and drop 129
- filters 132, 153, 154, 156
- loading member files 156
- opening or loading 129
- Project Properties Dialog 36
- Project Tools 36
- setting project configurations 36
- setting up project tools 142
- storing configuration settings with 135
- synchronize with external make-files/Workspaces 131
- use with version control 158
- Working Directory 133, 134, 154
- Prompt for Arguments 143
- Prompts
 - System 43
- PTG file 230
- PWB 415

Q

- Quick Search 215, 233
- quit
 - a long search 214
- Quoting 221, 222, 223

R

- Read User Makefile 130
- Read Visual Studio Workspace 130
- README.TXT 8, 18
- Read-Only Files 315
 - By File 316
 - By Type 316
 - Set upon Opening 317
- Rebuild 151, 161, 166
- Recompiling a DLL 397
- Recording a Keystroke Macro 302
- Redirect Output 143
- Reference Groups and Replacement Strings 222
- Regular Expressions 216, 220, 222, 223
 - Alternation and Grouping 221
 - Character Classes 219, 220

- Escape Sequences 218
- Escaping Characters in a Class 219
- Examples 223
- Iteration Qualifiers 220
- Matching End Of Line 221
- metacharacters 217
- Placing the Cursor 223
- positioning at Beginning/End of Line 221
- Reference Groups and Replacement Strings 222
- searching for more than one character 220
- special characters 217
- Reload 19
- Replace
 - Global replacement 211
 - Multiple Sources/files 205
 - Prompt options 211
 - Prompted replacement 211
 - Single replacement 211
- Replacement Options 210
- resolve name ambiguity 333
- resources
 - CodeSense parser priority 91
- Response File 146
 - Contents 146
- Restore Current Position 249
- Rich Text Editor 345, 346, 360
- Right Justify 22
- Run mode 356
- Running The Macro
 - AppBasic 355

S

- Sample EHTEST.CWB 356
- SAMPLE.DLL 392
- Save
 - All Files automatically 143
 - Current File automatically 143
 - Current Position 249
 - Files with Auto-Save 308
 - Large Files 324
- SC.DLL 391
- Scrap Buffers 108, 109
 - multiple 415

- Scrap Viewer 109
- scripting language
 - coloring 63
- Scroll Bars
 - Turning Off 321
- Scrollbars 45
- Search
 - for word at cursor 215
 - Ignore case 212
 - Maximal match 212
 - Multiple Sources/ Files 205
 - on toolbar 216
 - Regular expression 212
 - Restrict to selection 213
 - Retain selection 212
 - Select matching string 212
 - Whole word 212
- search
 - stopping or quitting 214
- Search After Go To 234
- Search and Replace
 - Dialog 204
- Search List 207
 - creating/modifying 209
 - Drive and Directory Lists 209
 - File Pattern 208
 - Include Directory 209
 - List Editing Buttons 209
 - Patterns List 209
- Search Menu 20
- Search multi-source
 - Documents only 206
 - Edit Modified Files 207
 - Fast Find 214
 - Files and folders 206
 - Output Window 208
 - Project 206
- Search Options 31, 204, 216
 - Case Sensitivity 21
 - Dialog 20, 204, 298
 - Prompt On Replacement 21
 - Regular Expressions 21
- Search Pattern List
 - editing 208
- Search results
 - search output window 205
- Search Subdirectories 207

- Searching
 - background Thread 208
 - binary/ hexadecimal data 225
 - CWFGREP 208
 - direction 210
 - Displaying and controlling Output 208
 - Exclude Start Position 213
 - incremental 215
 - listbox control 216
 - newlines 224
 - Regular Expressions 216
 - sets of files 207
 - spaces/tabs 224
 - Subdirectories 207
 - Wrap at beginning/end 213
- Searching Project Files 158
- SELECTION_LINE 341
- Selections
 - by Word 298
 - Closed 297
 - Line 297
- Selective Display 22, 226
 - preprocessor directives 227
 - save between CW sessions 228
- Send Mail 19
- Set As CWD 206
- SetStringMacro 147
- Setting Project Defaults 34, 124
- Shell Command
 - Tools Menu option 25
- Show Loaded Modules dialog 351
- Side-by-Side Differencing 256
- Smart Indenting 84
- Snippets
 - Adding a Snippet to CodeFolio tree directory 77
 - Adding or Removing a CodeFolio Snippet Directory 79
 - Code Snippet Properties Dialog 78
 - CodeFolio Tab on Project Window 29
 - CodeFolio Tab on the Project Window 74
 - Creating a Snippet from Clipboard or Document Selection 77
 - Deleting or Renaming CodeFolio

- Snippet 78
 - Directories 75
 - Editing a CodeFolio Snippet 78
 - Executing a Snippet into your Document 75
- Source Code Revision Control
 - Maintenance Dialog 170
- Spaces
 - setting 83
- Special Characters 69
- Special Keybindings
 - Appbasic 346
- speed
 - auto validating 307
 - CodeSense parser priority 91
 - increasing 135
- Spell Check
 - Tools Menu option 25
- Spell Check Dialog
 - Advanced Tab 271
 - Documents Tab 273
 - General Tab 268
 - Word Format Tab 269
- Spell Checking 268
 - Spell Check Dialog 268
- split window 46
- SrchSetFlags 396
- Standard Toolbar 30
 - Help button 31
 - repeat last search button 31
 - search for word under cursor button 31
- State File 282, 417, 420, 426
 - Contents 409, 418
 - Location 417
- Statements and Statement Blocks
 - API Macros 379
- Status Line Actions 298
- stderr 335
- stdin 335
- stdout 335
- stop
 - a long search 214
- StrAscii 384
- StrFileMatch 385
- StrFormatDate 385
- String functions
 - API Macros 384
- StringApnd 384
- StringCompare 384
- StringICompare 384
- StringLength 384
- StringNApnd 384
- StringNCompare 384
- StringNCompare 384
- StrToA 384
- StrLtrim 384
- StrSubStr 385
- StrTrim 385
- Submenus 284, 285
 - adding 284
- Swap Blocks
 - Changing the Number Allotted 319
- Symbol File 135
- Symbol lookups
 - CodeSense 96
- Symbol Parsers
 - making/creating 240
- Symbol Scanning 238
- Symbolic macros 146
- Symbols 229, 236, 247
 - Auto-Expand Collapse 239
 - Auto-Update Symbols database 135
 - Compute Document Symbols 135
 - configuring 240
 - displayed in symbols window 237
 - making/creating 240
 - navigating and using 238
 - sorting 239
- Symbols Parser 236
- Symbols Window 229, 236
- Synchronization
 - Borland C++ Builder File Setup 193
 - Borland C++ Setup 191
 - Delphi Setup 189
 - Initial Setup in CodeWright 185
 - Microsoft Visual Studio 187
 - MSVC++ Setup 187
 - Sending Menu Commands to Synchronized Environments 201
 - Supported Environments 185
 - TI Code Composer Setup 197
 - Visual Basic Setup 195

- Wizard 17
- syntax coloring
 - Adding a New File Type 64
 - adding keywords from a file 55
 - Comments 56
 - Creating a ChromaCoding Lexer 52
 - Creating a Language Support DLL 66
 - enable coloring 65
 - keywords 54
 - language specific editing features 65
 - Numbers 58
 - operators 54
 - preprocessors 54
 - stop coloring certain items 55
 - user-defined keywords 55
- SysEdit 294
- SysSetDefault 396
- SysSetFileLocking 410
- SysSetFlags 396
- system files
 - directories 133

T

- Tabs 220, 221, 222
 - controlling/customizing 62
 - insert spaces only 83
 - setting 83
 - showing tabs 46
- Tags 5, 235, 247, 427
 - Database 134
 - Setup 235
 - Support 235
 - Tagfind Function 236
 - TagNext 236
 - TagPrev 236
 - Using Database 236
- Tags Database 235
- Tags Support 229
- TagsWnn 229
- Technical Support 8
- Template Expansion 62, 67, 387
- Template macros
 - list of 72
- Templates
 - abbreviations that trigger 68
 - creating and modifying 68
 - example 69
 - ExtAssignTemplate 69
 - for language constructs 68
 - non-language-specific 69
 - run from Button Links 251
 - special characters in 69
 - Special Chars In 68
 - using macros in 69
 - viewing 68
- TERM.PM 343
- Texas Instruments Code Composer
 - Bi-directional synchronization 199
 - synchronization 197
- Text
 - Copy and Move 299
 - Expand/Collapse Sections of Text 301
- Text Drag and Drop 299
- Text Link DB 134
- Text Menu 22
- TI Code Composer Synchronization 197
 - Bi-directional synchronization 199
- Timestamps
 - Open tab-- Project Window 30
- Toolbar
 - Buttons 135
 - forward and back button 31
- Toolbar Search 31, 216, 298
- Toolbars
 - Adding and Changing Buttons 280
 - Adding New Toolbars 279
 - Appbasic 347
 - Auto-hide 277
 - Binding a Function to a Button 281
 - Customization 24
 - Customizing 279
 - Default Toolbars 275
 - Docking and moving 278
 - Docking Manually 278
 - Enabling and Disabling 278
 - Options 275
 - Standard toolbar 30
 - undock 27
- Toolbars dialog 347

- Tools
 - Custom 25
- Tools Menu 25
 - Adding External Program/Operation Links 287
 - API Command 39, 328
- Transformation
 - Fill Pattern 314
 - Override Pattern 314
- Transformation Patterns 314
- TransformFilename 385
- TYPEMAP 343

U

- Undo 20, 31
- Unicode
 - CodeSense translation 96
- Unindented Alignment 85
- UNIX
 - Changing EOL 404
 - Compiling from CodeWright 406
 - Enabling Auto-Sense File Type EOL 404
 - File Securities 406
 - File Transfer using FTP 318
 - Filename Case 406
 - Recommendations for Running CodeWright with UNIX 403
 - Using UNIX End of Line Characters 403
- Update Configuration Files checkbox 137
- UPDATE.TXT 18
- User-defined commands
 - Open tab--Project Window 30
- user-defined keywords 55
- UserDialog 347
- UserDialog Editor 347, 352
- Using the Tags Database 236
- Utilities
 - Supporting Files 388
 - Tools Menu options 25

V

- Variables
 - API Macros 366
- VCS tool 141
- VDOS 143
 - capturing output 166
- Version Control 132, 137, 169
 - Adding a New Command Line Provider 179
 - Adding CodeWright Project Files to an SCC Provider Project 173
 - Adding Version Control Files to a CodeWright Project 172
 - Associating Version Control Projects with CodeWright Projects 171
 - Configuring SCC Provider Support 183
 - Current Project Tree List 174
 - Customizing Version Control Commands 179
 - Default Command Line Version Control Commands 180
 - File Icons 174
 - Integration Methods 178
 - Location of the SCC Provider DLLs 184
 - Maintenance Dialog 170
 - Making Your Own Popup Menu 176
 - Menu 169
 - Modifying the Standard Popup Menu 175
 - Source Code Control Provider 183
 - Tools Menu option 25
 - User-defined Popup Menu 175
 - Using Command Line Version Control Provider 178
 - Using Multiple Configuration/Project Files 176
- version control utilities 137
- vi command set 295
- View Setups 24
 - Defaults 47
 - Synchronize text background 46

- Visibles tab 46
- View Setups Dialog 47
 - Colors tab 46
 - Font tab 46
 - General tab 46
- visibles
 - making visible 46
- Visual Basic Synchronization 195
- Visual Studio
 - Bi-directional synchronization 199
 - Synchronization 187
- Visual Studio Workspaces
 - creating CodeWright projects from 129
 - drag and drop to create project space 129, 300
 - open or load to create project space 129

W

- Web browser interface
 - turn on 101
- Web Page 8
 - Editing 101
 - turn on editing 101
 - turn on viewing 101
 - Viewing 100
 - WYSIWYG editor 100
- Where is it Defined? 333
- Wild Cards
 - File Filters 19
- Wildcard Patterns 19
- Win32 systems 344
- Window Menu 26
- Windows
 - Attributes 45
 - Creating with a Mouse 300
 - Difference between windows and documents 26
 - Docking and Moving 278
 - Docking Manually 278
 - move forward and back 31
 - split into multiple panes 46
- Windows and Documents
 - One Document Per window 43
- Windows API 112

Index

- Windows Explorer 300
- Wizards
 - configuration 17
- Word Selections 298
- Word Wrap 22
- Working Directory 133, 134, 408, 425
- Workspace
 - creating 156
 - defined 122
- Workspaces 121, 122
 - Closing Windows 158
 - creating 156
 - Deleting Buffers 158
 - loading 158
 - saving 156
 - saving automatically 157
 - updating current 158
- WYSIWYG 101
- WYSIWYG HTML
 - turn on 101
- WYSIWYG HTML editing 100

X

- XML split window viewer 103
- XSUBPP.PL 343